THE BEST SELLING COMPUTER PROJECTS MAGAZINE JANUARY 1985

ELECTRONICS & COMPUTING MONTHLY

AN EMAP PUBLICATION

USA $2.95
Germany D6.00
Singapore S$4.95
95p

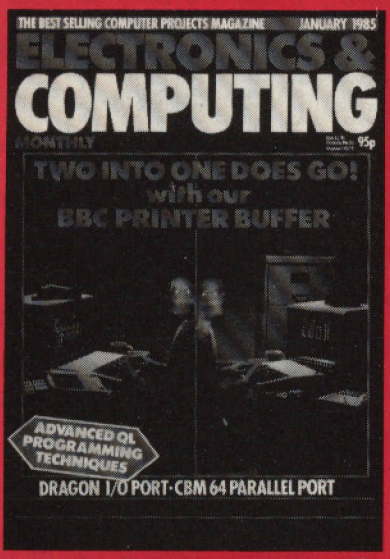
TWO INTO ONE DOES GO! with our BBC PRINTER BUFFER

ADVANCED QL PROGRAMMING TECHNIQUES

DRAGON I/O PORT · CBM 64 PARALLEL PORT

SPEEDY BBC EPROM BLOWER

SPECTRUM WAFADRIVE-MICRODRIVE ALTERNATIVE?



COVER PHOTO BY ROB BRIMSON

ELECTRONICS & COMPUTING Contents

Vol 5 Issue 1

PROJECTS

FEATURES

REVIEWS

PLUS

Plus within *Your Robot*
Beasty shape cutter – a novel application of Commotion's servo control system.
Remcon Teach-Robot – setting the scene for next month's DIY robot builders guide.

Electronics & Computing Monthly
Scriptor Court, 155 Farringdon Road,
London, EC1R 3AD

Editorial 01-833-0846
Editor Gary Evans
Deputy Editor William Owen
Production Editor Liz Gregory
Administration Serena Hadley

Advertising 01-833-0531
Advertisement Manager Anthony Herman
Chief Executive Richard Jansz
Classified Tracey Keighley
Advertising Production Yvonne Moyser

Production 01-833-0531
Art Editor Jeremy Webb
Make-up Time Graphics
Publisher Alfred Rolington

Distribution
EMAP National Publications

Published by
EMAP Business and Computer Publications

Printed by
Riverside Press, England

Subscriptions
Electronics & Computing Monthly, (Subscription Department), Competition House, Farndon Road, Market Harborough, Leicestershire.

Electronics & Computing Monthly is normally published on the 13th day of each month.

Speech therapy

Most speech synthesisers which make use of allophones allow the user to create any English words (or foreign for that matter) that they require for a particular application rather than restricting the range of words to a limited vocabulary supplied by the manufacturer of the system. However, there is one drawback. This is the fact that it can be a time consuming and frustrating exercise to select the right sequence of allophones necessary to produce a convincing pronounciation of any particular word.

Users of the Currah Microspeech who find themselves with this problem will be pleased to learn that the company are now able to supply a dictionary showing how 2140 commonly used words may be broken into their allophone components. The dictionary also shows the construction of 590 Christian names and 250 'games' words. Thus we are told that TOTAL should be entered as

(tt)(eau)(tt)u(ll)

Not a lot of people know that and it could take a great deal of experimentation before that particular series of allophones was discovered. In addition any word that is not in the dictionary may be constructed by using the required section of a similar word that is already in it. Thus the word ACE could be formed by selecting the (ay)ss part of the word TRACE-(tt)r(ay)ss.

The dictionary costs a reasonable, in view of the amount of work that must have gone into its production, £3.50 inc. p&p. It is available from the sales department at Currah Computer Components Ltd., Hollymount, Woller Road, Hartlepool, Cleveland.

Please note

In the article describing a DataPad for the BBC micro in last month's issue we drew attention to a number of errors in the software listing and printed a list of corrections. The listing was however correct as published and no alterations need be made for the correct operation of the project.

For readers who prefer not to type in the software, the authors of the article can supply the program on cassette (easily transferable to disk) at a cost of £3.75 inclusive of post and packing. Send orders to Ian Walker, Woodleighsoft, 13 Woodleigh, Walton, Brampton, Cumbria, CA8 2DS.

In our Comms Column, the number shown for the BBC's Micro Live Bulletin Board was incorrect. The number is 579 2288. Apologies for any inconvenience caused by the error.

When Markets Merge

The distinction between computers generally regarded as home machines (the three Sinclair models, the Commodore trio and the BBC micro all fall into this market) and those computers that attract the label of Personal Computer (PC) is becoming increasingly blurred. Apart from one or two notable exceptions, however, the media and the distribution trade are failing to recognise this coming together of the two ends of the microcomputer sphere.

Only a short time ago the PC and the home markets were characterised by significant differences in price and performance. PCs were 16-bit machines that sold for in excess of £2000. Home micros were based on 8-bit micros and sold for hundreds rather than thousands of pounds. Today though the division between the two market sectors is far from clear cut.

The QL is a good example of a computer that bridges the gap between the home and PC (business) areas. A complete QL system, computer, printer and monitor comfortably breaks the £1000 barrier yet offers a level of performance that more than satisfies the requirements of the business user. Price reductions in hardware from the likes of Hitachi and Sanyo also mean that full disk-based 16-bit computers are available for around £1000.

£1000 is the magic figure because whatever computer forms the heart of the system, whether it be an 8 or 16 bit machine and whether it costs £300 or £500 by the time the price of a dual disk drive, printer and monitor is taken into account, the user is faced with a bill that is in the region of that magic £1000.

As an increasing number of micro users upgrade their existing systems and new users enter the field by way of the more sophisticated computers on sale today, the media in this country will have to address themselves to the changing nature of the market. In the States, Byte, one of the first US computer magazines, manages to draw together editorial covering a broad spectrum of the market. The publishers have the advantage of a large number of editorial pages and to a certain extent can afford to satisfy the wide range of interests represented by owners of say the Radio Shack (Tandy) CoCo at one end and the Mackintosh at the other.

Many magazines in this country do not have the luxury of hundreds of pages and have to restrict their coverage to a narrow band of the market, a process which can take one of two forms. A title can take a horizontal slice of the market and concern itself with a number of machines that are very similar in price and performance. Alternatively the slice can be vertical resulting in a magazine aimed at the users of one specific computer, the so-called user title.

While publishers are often forced to take one or other of these courses of action it seems that neither is in the best interests of the reader or the industry. It is back to the specialist, rather than generalist approach that typifies so much of the education in this country. In order to get the most from computing it must surely be in peoples' interests to read about a brand range of topics. An article that is superficially aimed at users of the Spectrum could contain ideas useful to owners of a BBC system complete with Z80 second processor. Interfacing ideas for a particular computer could well find applications in a wide range of other manufacturers' hardware.

It is only by reading across as wide a range of material as possible that owners of microcomputers can expect to get the most of their machine and to keep up with the latest techniques and news from the industry. With this in mind as *E&CM* enters the 1985 publishing year you will notice an expansion of our coverage. We're not planning an A/D convertor for the IBM PC as yet, but space permitting, you will notice items on computers that are not as yet covered in the majority of consumer computer magazines. It remains to be seen whether other 'home' computer magazines will recognise this change in the market and broaden the scope of their coverage. But as we've always preferred to lead rather than follow we hope that you will agree with our decision.

Gary Evans

Skywave expand

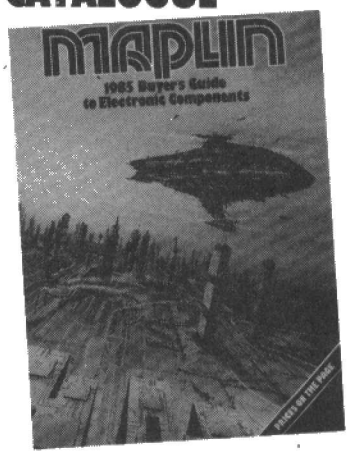
Skywave, best known for their commitment to spreading the FORTH word via a range of EPROM-based versions of the language for the popular home micros, are moving into the supply of hardware add-ons. An expandable RS232 interface for the Amstrad CPC464 is an example of the company's growing range of products.

The interface features a built-in crystal controlled baud-rate generator that gives the choice of eight data transmission rates between 75 and 9600 baud. Based on the 8251 IC the interface supports operation at different transmit and receive rates thus making it compatible with Prestel modems.

Provision is made for future expansion by way of a sideways ROM socket that could contain software to provide more advanced RS232 features. In addition expansion cards to allow the use of seven sideways ROMs or seven 8255 based parallel ports (giving a total of 168 parallel bits of output) can also be fitted.

The interface costs £59 excluding VAT with an extra £3 p&p from Skywave Software at 73 Curzon Road, Boscombe, Bournemouth, BH1 4PW.

BUMPER COMPONENT CATALOGUE

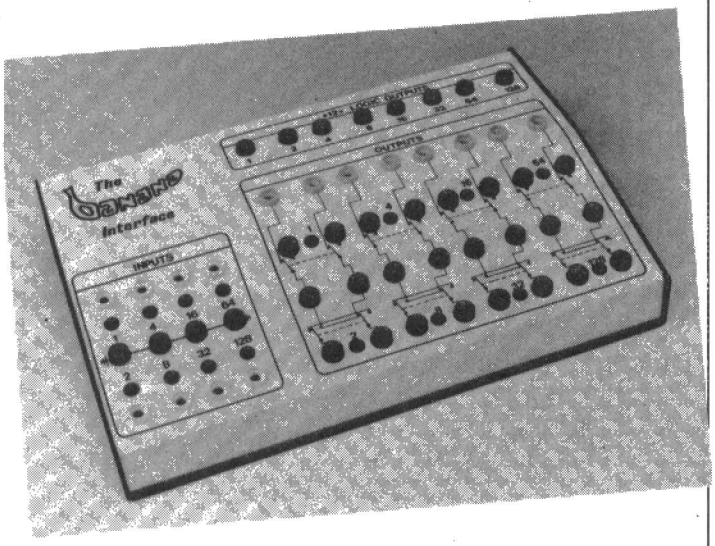


The new Maplin catalogue is now available in a large number of newsagents throughout the country. The new catalogue is one of the most comprehensive guides to components and kits available and is a must for any serious constructor.

The catalogue costs just £1.35 from newsagents but in case your local does not have a copy it is available from the company's Rayleigh warehouse. Send £1.75 (the extra 40p is to cover postage on the weighty tome) to Maplin at PO Box 3, Rayleigh, Essex, SS6 8LR quoting order reference CA02C.

NEWS NEWS NEWS NEWS

Versatile Banana interface



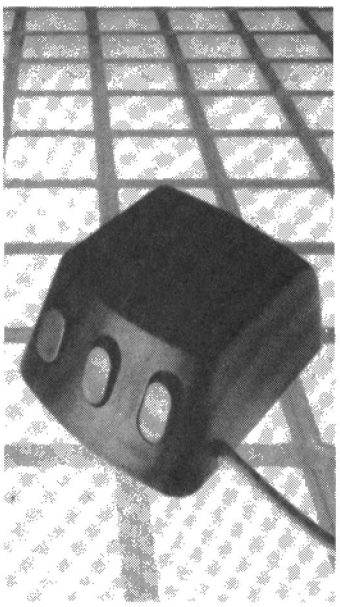
A versatile interface, going under the unlikely name of Banana, is now available for the BBC micro, the VIC 20 and the CBM 64. Described as virtually bomb proof, the unit has been designed so as to appeal to educational establishments throughout the country.

The list of features include eight independent outputs driving relays and 8, +12 volt logic signals for controlling DC motors, stepper motors and other devices. Independent control of up to 8 motors in ON/OFF mode or 4 motors in FORWARD/REVERSE mode is possible. Inbuilt stopping circuits are incorporated. Using this facility, can bring small DC motors up to the accuracy of Stepper motors. Eight independent inputs ensure intelligent control of motors and other devices using suitable sensors.

An anti-panic circuit is incorporated so that on BREAK or RUN/STOP commands all the functions are inoperative. A full range of specific experiment hardware and software will be available.

For further details contact:
Peter R. Bull
Sales and Commercial or
Peter Brierley – Technical on
0723 584250

AMX MOUSE BRINGS ICONS TO THE BBC



BBC micro owners wishing to keep up with the Jones' must now add another item to their shopping list. The introduction, by AMX, of a mouse designed for the machine means that all the fashionable user friendly facilities that this add-on can provide are now available to BBC owners.

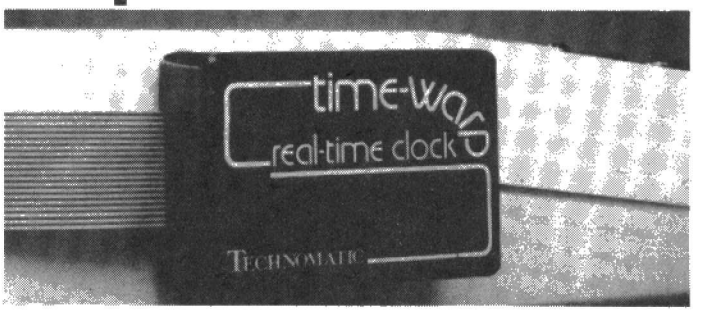
The AMX mouse simply connects to the micro's user port, software for the system resides in a sideways ROM.

The ROM based software allows programs to be written in either BASIC or assembly language to incorporate features such as icons, windows and pointers. Three user buttons on the mouse may be individually configured by the user but normally perform the operations of EXECUTE, CANCEL and MOVE.

Demonstration software supplied with the mouse includes an advanced drawing program which shows the full potential of the combination of the AMX mouse and the potential of the BBC's graphics facilities.

Full details from Advanced Memory Systems at Green Lane, Appleton, Warringtron, WA4 5NG. Telephone 0925 602690.

Technomatic enter Time Warp



Technomatic's Time Warp Real Time Clock Calendar for the BBC micro brings all the facilities of a RTC system yet costs only £29 plus VAT.

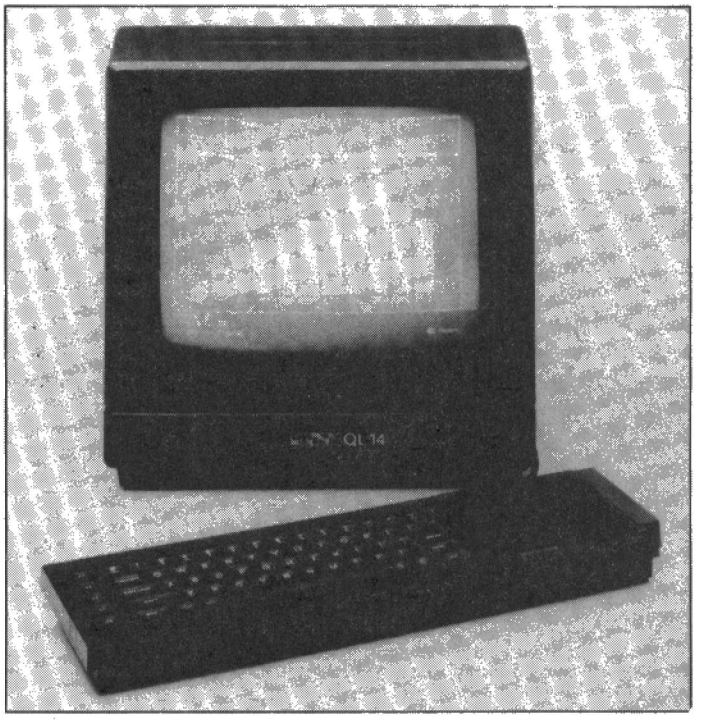
The battery backed design offers a range of facilities that include a continuous display of time at the top of the screen, an electronic diary with an auto alarm that can give reminders of appointments and the time and date stamping of files.

Installation is straightforward – a single connection to the Beeb's user port provides the unit with all its data and power requirements. The system does not require the use of yet another sideways ROM.

The Time Warp RTC is supplied complete with instruction manual and cassette-based software that may easily be transferred to disk. Software supplied includes CLK, a demonstration program, written in assembly language that shows how time and data may be continuously displayed on top of a mode 7 screen while another program is run simultaneously.

The Time Warp Real Time Clock Calendar is available from Technomatic at 17 Burnley Road, London, NW10 1ED.

Sinclair QL monitor



For many microcomputer owners the cost of a monitor is often not possible to justify as the quality of their computer's video display generator is such that an ordinary TV will give quite adequate results. QL owners wishing to get the most from the machine must however splash out on a dedicated monitor. Prism have just announced that they are to distribute just such a product which will go under the name of QL14. This 14" model is to sell for just £199.99 thus undercutting most of its rivals by a considerable margin.

The QL14 allows the full 85 column width of the QL to be displayed and is supplied with a cable that connects the monitor to the computer's RGB socket.

SPEEDY BBC EPROM BLOWER

This low-cost, easy to build device is a blower with a difference. Andy Green presents a design which programs twice as fast as others and which also features an on-board converter.

If the prospect of *E&CM* publishing an EPROM blower for the BBC micro instills a feeling of déjà vu, then you may be forgiven for we have in the past presented a design for just such a project. However, this design has a number of unique features that make it of special interest. These include an intelligent programming algorithm that means this blower is able to program twice as fast as many 'conventional' designs, an onboard converter that is able to generate the 21V programming pulse from the Beeb's 5V rail and it makes provision for the ability to program both 2764 and 27128 devices. Perhaps the most important aspect of all to some constructors is the fact that its 41C design is both low-cost and easy to build.

We have a problem

A major problem that had to be overcome during the design of the project was how the 14 address lines, 8 bi-directional data lines and 6 control lines required by the programmer could be derived from the 10 I/O lines provided by the BBC micro's user port. The first difficulty was overcome by the use of a 4040 12-bit counter. This does the job of 12 of the 14 address lines with just two. One of these is used to reset the counter and the other to increment the IC by way of its CLK input.

The next step in reducing the I/O overhead was to multiplex 8 of the user ports' lines with IC2, a 74LS374 8-bit data latch. The 9th bit of the user port (CB2) is used to strobe data into the latch. Note that if CB2 does not get strobed, the '374 will ignore all data and will continue to output the last valid data.

As a result of the multiplexing process, an extra eight bits of data are available. Two of these are required to control the 4040 counter as described above, another two supply address lines A12 and A13; the use to which the remaining four lines are put will be covered later.

The eight data lines of the user port are also connected to the eight data pins of the EPROM. In theory this should mean that the programmer can read/write data to the EPROM and only send new data to the latch when it strobes CB2. Unfortunately this is not the case. When the EPROM is being read from it is, from necessity, outputting data. The catch is that OE, the EPROM input that controls such outputs, is controlled from the latch. Thus while the latch can be set easily enough to make the EPROM output a byte at a particular address, the problem is that when the EPROM is outputting data to the bus it is not possible to provide the latch with a valid strobe.

The answer to the problem is to deselect the EPROM whenever the latch strobe goes (active) low. Because it is the rising edge of the strobe line that will latch the new data, it is in any case necessary to pull the strobe line low in order to latch new data. Q1 inverts the strobe and takes this inverted signal to the CS input of the EPROM.

IC3 is a 6mS pulse generator with an output which is normally high but which goes low for a period of about 6mS upon receipt of a rising edge at its input. Its function is to derive the PGM pulse that controls programming of the EPROM. Capacitor C2 ensures that the 6mS pulse starts just after the start edge is presented to the IC; this is to give the computer enough time to get the right data onto the data lines.

The output of the pulse generator is also fed to CB1, the in line of the computer, in order that it 'knows' when the pulse has started/ended.

Those of you familiar with the programming requirements of EPROMs may wonder why we are talking of a 6mS programming pulse. Surely 50mS is a figure that seems more familiar than 6. The answer to this conflict lies in the traditionally cautious way in which manufacturers specify various aspects of their wares. The 50mS programming time refers to the "worst case' condition.

An erased EPROM contains all 1s (&FF) and to change any of its FF bytes to 00 bytes does indeed require about 50mS of programming effort. If, however, it was required to change only one bit of the FF byte then only a fraction of this time would be required. For once the logical idea that ⅛ of the 50mS period would suit is correct – thus the derivation of our 6mS fundamental programming unit.

The programmer, before programming a byte, will calculate how many changes to zero are called for. It will then burn the byte into the EPROM for that many periods of 6mS (any FF bytes are skipped over).

Casual calculations.

As an aside, we present **Listing 1** which will count up the number of 1 and 0 bytes in, say, Wordwise. They turn out to be about equal (for those who like to collect such trivial information Wordwise has 38,638 0 bits, that is 59% and 26,898 1 bits, 41% of the total).

As previously mentioned one of the unique features of this project is the onboard 21V power supply derived from the 5V rail of the BBC micro. This minor miracle is achieved with the TL497A switching regulator from TI with a supporting cast of a few resistors and capacitors plus an inductor.

The 21V output of the regulator does not go straight to the Vpp pin of the EPROM – it first goes through Q2/3. This circuit, courtesy of Brian Alderwick and Peter Simpson, switches the 21V line to the EPROM when a control line is high. When the 21V line is 'switched off' diode D1 supplies 5V instead. This means that the Vpp pin of the EPROM sees either 21V when programming or 5V 'normally'. The bottom line is that the 21V line is switched to the EPROM under computer control.

The above description highlights a caution. When the computer is first powered up, all functions controlled by IC2, the '374 latch, will be undefined. This means that

LISTING 1.

```
   10 REM "Counts the number of 1/0 bits
 in the area &3000-&5000
   20 REM
   90 lo=&70:zero=&72:ones=&76
  100 FORY%=0TO2STEP2:P%=&C00:COPTY%
  110    .count LDA#&30:STA&71:LDY#0:STY&
70
  120    .c1 LDA(lo),Y:LDX#8
  130    .c2 ROR A:BCCc3:INCones:BNEc4:IN
Cones+1:BNEc4:INCones+2:BNEc4
  140    .c3 INCzero:BNEc4:INCzero+1:BNEc
4:INCzero+2
  150    .c4 DEX:BNEc2
  160    INY:BNEc1:INC&71:LDA&71:CMP#&50:
BNEc1:RTS
  200    ]:NEXT:!zero=0:!ones=0:PRINT"Sta
rted:":CALLcount
  210 O=!ones:Z=!zero:Op=INT((O/(Z+O))*1
00+.5):Zp=INT((Z/(O+Z))*100+.5):PRINT"Ze
ro: ";Zp;"%, Ones: ";Op;"%"
```

there is an even chance that the 21V line will be applied to the EPROM. For this reason, leave the ZIF socket empty until the control program that sets up the initial conditions of the programmer has been run.

A LED controlled by the latch is an indication that programming is taking place.

Getting it together

There is no *right way* of constructing a project but here are a few personal preferences. Firstly, install the IC sockets (it is strongly recommended that sockets are used) but do not insert the ICs. Next solder in the larger components and the smaller items.

When it comes to the connection of the ribbon cable, **Figure 2** is a must. The strands of the cable should be cut down about 1 centimetre and the ends tinned. Next the row of 12 holes above IC2 should be located. One end of these holes is marked 'R' on the PCB for the project. This is home for the red strand on the cable. The others follow in a logical order with the exception of the last two which swap places.

The other end of the cable is connected to a 20 way IDC connector, the only point to note here is that the red strand goes in the end of the connector marked with a small triangle – this designates pin 1 of the connector.

A 100nF capacitor (C5) should then be soldered between pin 1 and 20 of IC2 and its body bent back over the ribbon cable. This acts as a decoupling capacitor and prevents any potentially embarrassing transients from making themselves felt when the EPROM is inserted/removed.

The PCB also makes provision for a cable clamp to be clamped to the ribbon cable.

After a final check of the board insert all the ICs paying particular attention to the health of IC1.

To the test

When construction is complete, the program shown in **Listing 2** allows correct operation of the programmer to be verified. The program tests the various sections of the project. These tests include:

1) Flash programming LED. If this test

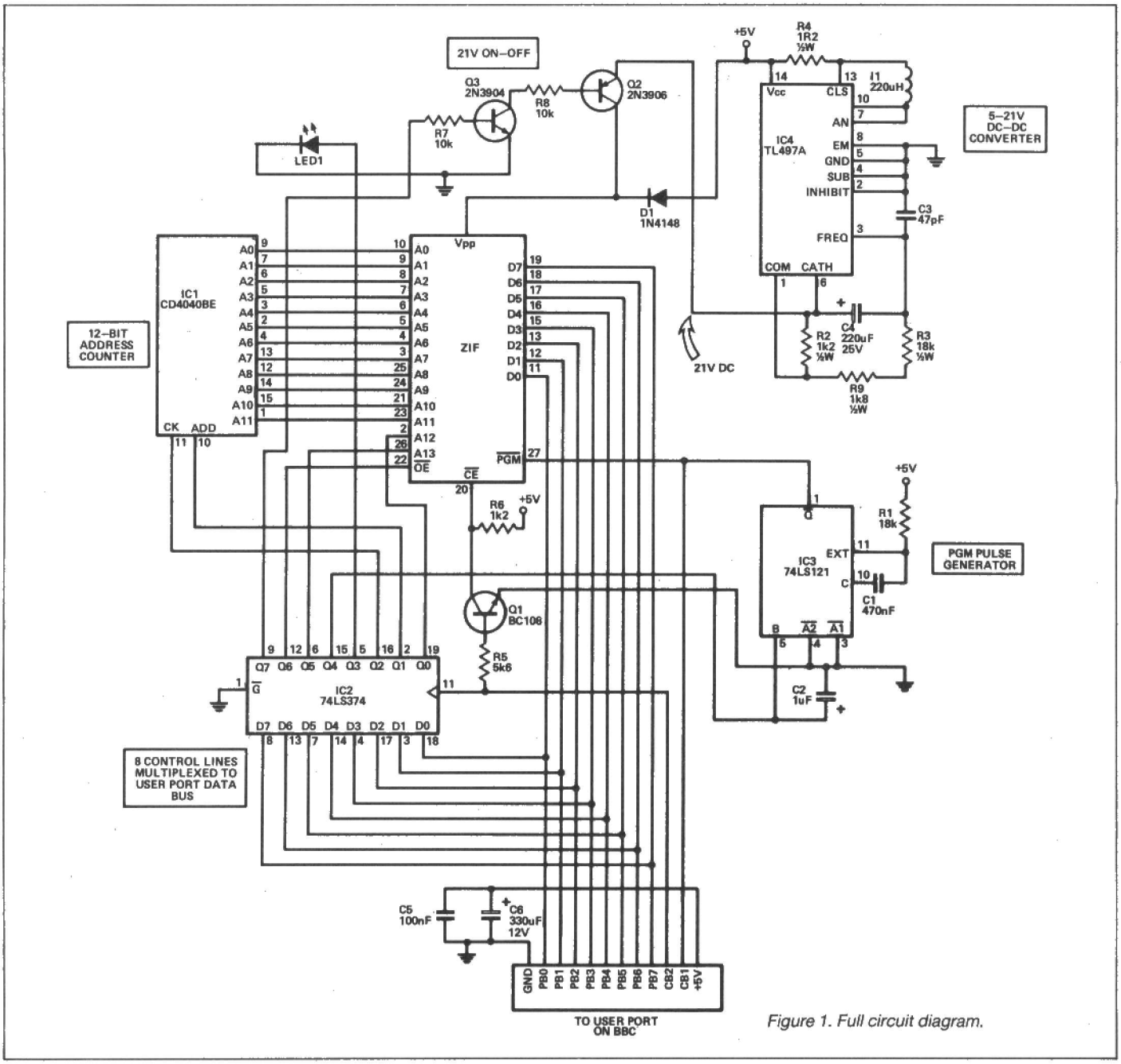


Figure 1. Full circuit diagram.

fails either the LED is in the wrong way round or there is something wrong with the latch. This test is one of the most important as correct operation of the latch is vital for the programmer to function. Check that the latch is in the right way round and check for any solder splashes on the board.

2) 21V on/off. Pin 1 of the ZIF socket should toggle between 21V and 5V once every 5 seconds. If not check for 21V at the + side of C4. If the 21V is present at this point then the TL497 is operating correctly and the fault lies in the area of Q2/3. Also check the orientation of idode D1.

3) Pulse length check. This measures both the PGM pulse and the delay between the pulse request and its generation. The delay should be around 20uS and the pulse about 6mS. If not, check IC3 and associated components.

4) Address line test. This checks for correct operation of all address lines. The program pulses each address line in turn and reads back the result. It then gives a verdict on each location. If there is any failure during this section of the program check the tracks from the pin that failed for solder splashes etc.

Next month

Full details of all the software needed to drive our super-speedy BBC EPROM blower.

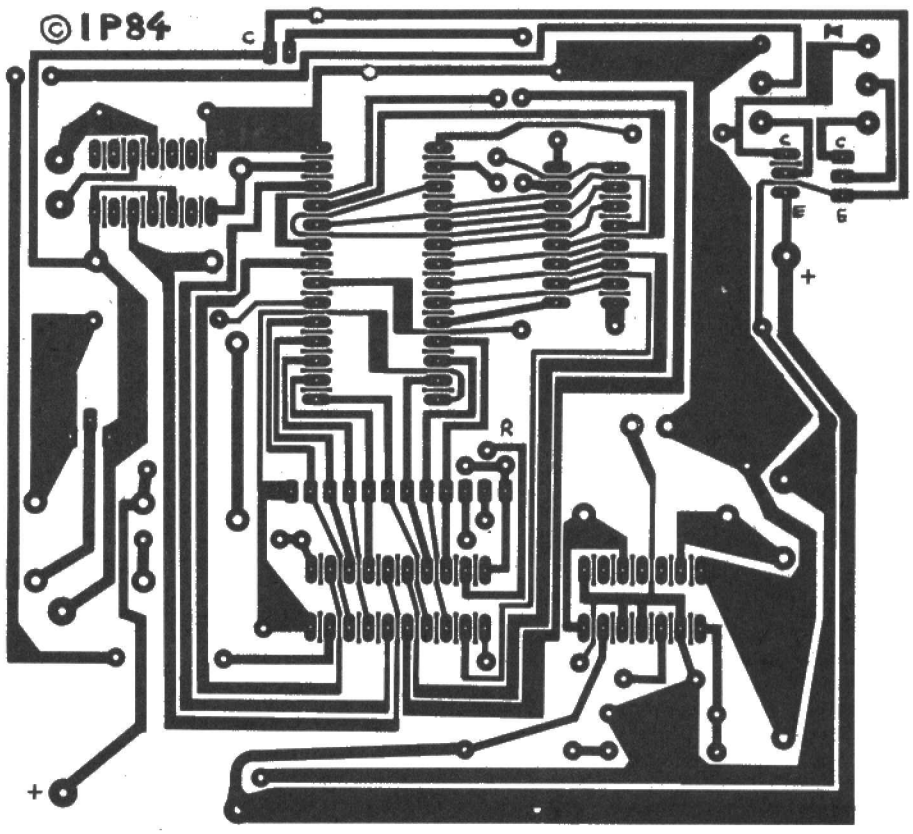


Figure 3. Foil pattern.

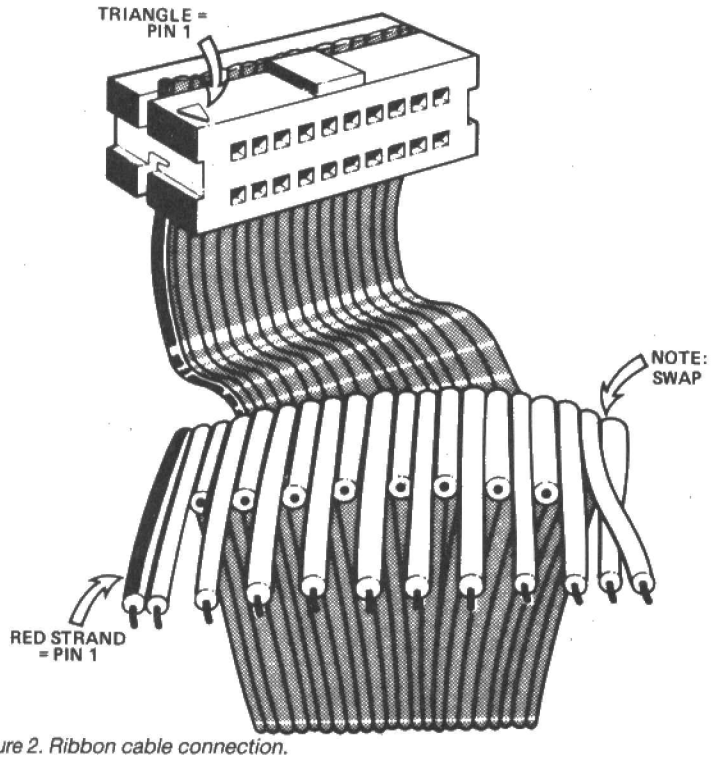


Figure 2. Ribbon cable connection.

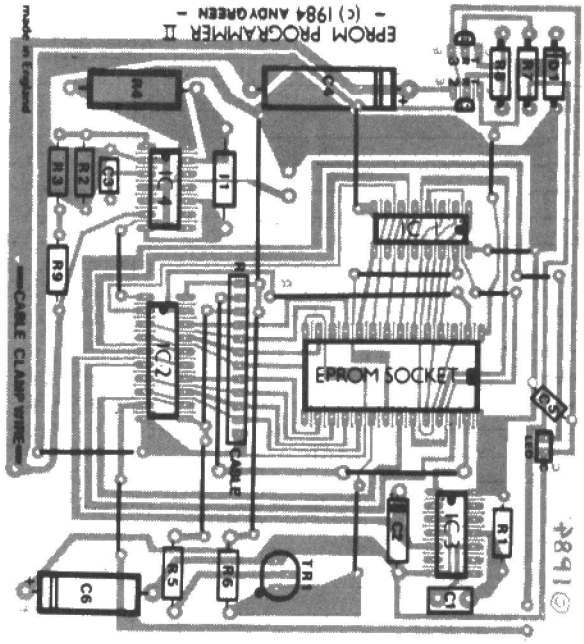


Figure 4. Overlay for EPROM programmer.

Parts List

IC1	CD4040BE	R1	18K	R9	1K8 ½W	C6	330nF@12V submin	TR1	BC108	1	28pin ZIF
IC2	74LS374	R2	1K2 ½W					TR2	2N3906		
IC3	74LS121	R3	18K ½W	C1	470ˆF			TR3	2N3904	2	x 14pin DIL skFr
IC4	TL497A	R4	1Ω2 ½W	C2	1pF	D1	IN4148				
		R5	5K6	C3	47pF			I1	220μH	1	x 16pin DIL skFr
		R6	1K2	C4	220μF@25V	LED1	Submin Low Current	Im	20-way Ribbon Cable	1	x 20pin DIL skFr
		R7	10K	C5	100ˆf			1	20 way IDC skt		
		R8	10K								

LISTING 2.

```
    10  REM "Eprom Programmer Tester
    15 DIM F% 200
    20 ads=&70:result=&72
    30 PROCasm:MODE7:CALLtpld
   100 PROCtestLED:PROCtest21V:PROCtest74
121:PROCadstest:RESTORE:MODE7:PRINT"THAT
'S ALL THE TESTS.  HIT ANY KEY TO   REST
ART.":A=GET:GOTO100
   900 DEFPROCpage(A$):CLS:A$=CHR$132+CHR
$157+CHR$135+CHR$141+STRING$((34-LENA$)D
IV2," ")+A$:PRINTA$'A$;TAB(0,24);CHR$131
;CHR$157;CHR$132;"    Press <space> for
next test";TAB(0,4);:ENDPROC
  1000 DEFPROCtest74121:PROCpage("74LS121
 PGM- Pulse Generator"):PRINT"IC3 genera
tes PGM- pulses for the EPROM during pro
gramming.  These must be about6ms long (
+/- 0.5ms), and have to begin 20 - 100 u
sec after they've been asked  for."
  1002 PRINT'':CALLtpld:IF(?result)=0THEN
PRINT"¹NO RESPONSE":GOTO1015  ELSE PRINT
"²Start Delayed by :";:D%=((!ads)AND&FFF
F)-9:PRINT;D%;" usec"
  1005 A%=INKEY(5):CALLT121:IF(?result)=0
THENPRINT"¹NO RESPONSE":GOTO1015 ELSE PR
INT"²Pulse length      :";:D%=(((!ads)AND
&FFFF)-D%-9):PRINT;D%/1000;" ms"
  1010 PRINT''"⁵Average program time for
2764 (8K) :"''TAB(12);:T%=(32768*D%)/1E6
:PRINT;T%DIV60;" mins  ";T%MOD60" secs"
  1015 REPEAT:UNTILINKEY(0)=32:ENDPROC
  1020 DEFPROCtestLED:PROCpage("LED TEST"
):PRINT"The LED should be flashing aroun
d twice a second.  If it isn't check tha
t the   LED is the right way around."
  1022 REPEAT:A%=8:CALLlatch:PRINTTAB(19,
13);"³";CHR$255:A%=INKEY(15):IFA%<>32THE
NA%=0:CALLlatch:PRINTTAB(19,13);"  ":A%=
INKEY(15):IFA%<>32THENUNTILØ
  1025 GOTO1040
  1030 DEFPROCtest21V:PROCpage("21V Trans
istor Switch Test"):PRINT"Pin 1 on the Z
IF should be alternating  between 5V and
 21V every five seconds.  The LED should
 be lighting when pin 1 isat 21V."
  1035 REPEAT:A%=&88:CALLlatch:PRINTTAB(1
8,13);"³21V";:A%=INKEY(500):IFA%<>32THEN
A%=0:CALLlatch:PRINTTAB(18,13);"15 V";:A
%=INKEY(500):IFA%<>32THENUNTILØ
  1040 A%=0:CALLlatch:ENDPROC
  1050 DEFPROCadstest:PROCpage("Address L
ine test"):PRINT"Use a couple of inches
of wire to  link pin 18 on the ZIF socke
t to other ZIF   pins.  When connected h
it <space>"
  1055 A%=0:CALLlatch:A%=4:CALLlatch:C%=0
:FORT%=0TO13:READB%:PRINT'"³A";T%;TAB(5)
;": Link pin 18 to pin ";B%;
  1060   PRINT;TAB(30);:A%=GET:?&FE62=0:R
%=?&FE60
  1070   IFT%=0THENA%=0:ELSE A%=(2^(T%-1)
)-1
  1080   C%=C%+A%+1:CALLchng:?&FE62=0:A%=
INKEY(1):S%=(?&FE60)AND64:IF(R%=&BF)AND(
S%=64)THENPRINT"²OK"; ELSE IF(R%=&BF)AND
(S%=0)THENPRINT"¹short";:ELSE PRINT"¹ope
n";
  1090   NEXT:GOTO1015
  1990 DATA10,9,8,7,6,5,4,3,25,24,21,23,2
,26
  1995 DATA11,12,13,15,16,17,18,19
  2000 DEFPROCasm:FORY%=0TO3STEP2:P%=F%:[
OPTY%:.T121 SEI:LDA#0:JSRlatch:LDA#&10:J
SRlatch
  2005   .time LDA#0:STAads:STAads+1:LDA#
&FF:STA&FE64:STA&FE65:LDA#&10:.t1 BIT&FE
6D:BNEt1a:INCads:BNEt1:INCads+1:BNEt1:LD
A#0:.t1a STAresult:LDA&FE64:EOR#&FF:STAa
ds:LDA&FE65:EOR#&FF:STAads+1:CLI:LDA#0:J
MPlatch
  2007   .tpld SEI:LDA#0:JSRlatch:LDA#&10
:STA&FE60:LDA#&CE:STA&FE6C:LDA#&EE:STA&F
E6C:JMPtime
  2010   .chng LDA#0:JSRlatch:LDA#2:JSRla
tch:DEC&404:LDA&404:CMP#&FF:BNEchng:DEC&
405:LDA&405:CMP#&FF:BNEchng:LDA&40D:LSRA
:LSRA:LSRA:LSRA:AND#1:STAresult:LDA&40D:
AND#&20:ORAresult:JMPlatch
  2500   .latch PHA:LDA#&FF:STA&FE62:PLA:
STA&FE60:LDA#&CE:STA&FE6C:LDA#&FE:STA&FE
6C:RTS
  2999   ]:NEXT:ENDPROC
30000 INPUT"Which socket (0-15): " A%:P%
=&C00:[OPT2:STA&FE30:LDA#&80:STA&71:LDA#
&30:STA&73:LDA#0:STA&70:STA&72:LDY#0:.L
LDA(&70),Y:STA(&72),Y:INY:BNEL:INC&71:IN
C&73:LDA&71:CMP#&C0:BNEL:LDA&F4:STA&FE30
:RTS:]:CALL&C00
>
```

DRAGON I/O PORT

Despite being 6809 based, the Dragon 32 has no user port. Chris Walcot puts this to rights with a device which will plug into the ROM port and provides software for input and output control.

Although the Dragon 32 is based on the very powerful 6809 microprocessor, the machine unfortunately has no user port for the hobbyist. In this project, *E&CM* show how this can be overcome quite simply with this inexpensive I/O port which plugs into the Dragon ROM port. With the software provided the port can easily be controlled for both input and output.

Circuit diagram

The design is based upon Motorola's powerful 6821 I/O chip. The memory address decoding is carried out by the 74LS27 and the 74LS30 logic chips (as shown in **Figure 1**) to locate the 6821 between memory addresses 52960 and 52963. The three outputs from these two chips go to pins 22 to 24 which enable the 6821. The two lowest address lines (A0 and A1) go to pins 36 and 35 to select ports A or B. Pins 26 to 33, on the 6821, accept the eight data lines from the Dragon data bus. The overall circuit diagram is shown in **Figure 2.**

The 6821 has two 8 bit ports and four control lines., Addresses 52960 and 52961 are used to control and read port A, while 52962 and 52963 likewise control and read port B. Actual programming techniques are handled by the program shown in **Listing 1.** Alternatively, if you wish to develop your own customised software, more detail can be found in the 6821's data sheet.

Input and output

Input is really quite simple. If one of the BITs is connected to +5V it will be logic 1, whereas if you connect them to 0V it will be logic 0. Similarly it is just as easy to drive an external circuit via the interface. Each bit can easily trigger up to 5V at 20mA. **Figure 3** shows how simple it is to drive a LED. A relay or a different circuit may also be driven.

Construction

It is much easier and cheaper we decided to build the circuit on a PCB rather than employ vero board and have to use lots of cables and connectors which tend to cost the earth. Once you have obtained the PCB the vero pins should be inserted, followed by the IC sockets. The next step is to solder the PCB connectors to the board. Finally the ICs must be inserted into their sockets as indicated in **Figure 4.**

Before trying out the interface it is better to make sure none of the PCB tracks have solder bridges on them because these could prove disastrous for your computer. Once all these steps have been carried out, turn off your Dragon and plug the interface (component side up) into the ROM socket on the right-hand side of the computer. Turn the computer on and wait for the usual display to appear. If this does not happen within a few seconds, then immediately switch off your computer for further investigation!

"Design is based upon Motorola's 6821 I/O chip".

Testing

Once the computer's normal screen is shown, load the software into the computer. To fully test the new port first connect PA0-PA7 (**Figure 4** shows their position) to 0V. Next RUN the software and select all inputs by entering '0' into the two ports and pressing 'F' when this is completed. The screen will clear and a new set up will be shown. The upper half of this shows the state of ports A & B (for input), while the lower half of the screen is used for output like the first screen. Below "PORT A'S CONDITION" you should see a row of eight "0"s appear.

If you now connect all 8 bits to +5V and then press "F", a row of eight "1"s should be displayed where the noughts were.

Testing output is more difficult because you need an external circuit to display the ports' condition. The simple circuit shown

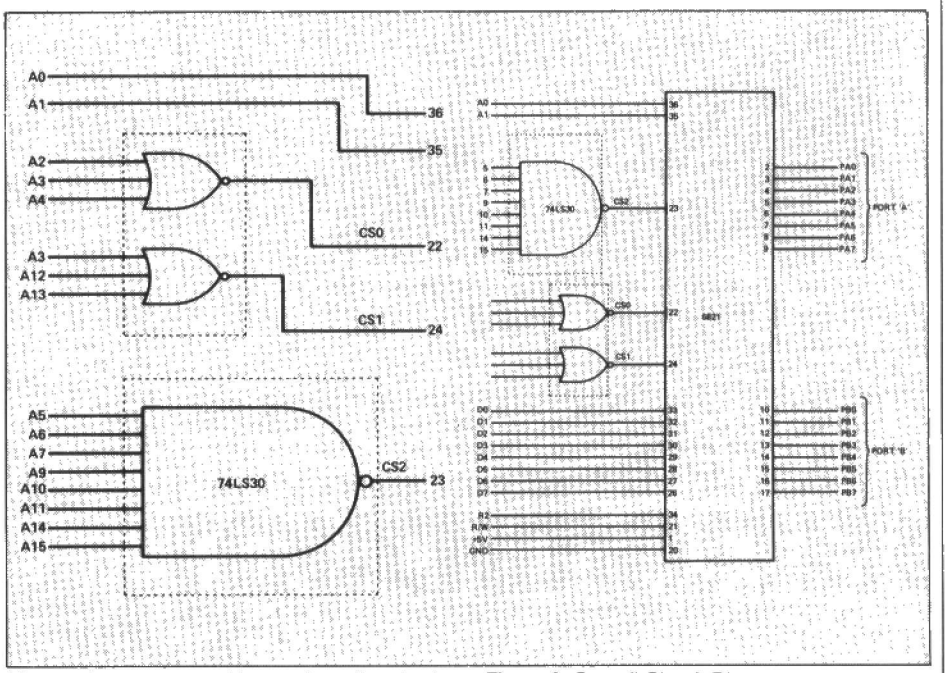
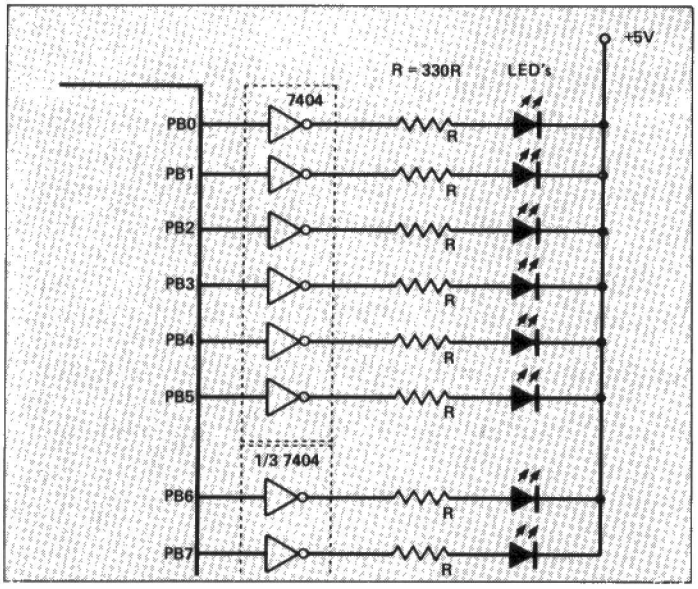
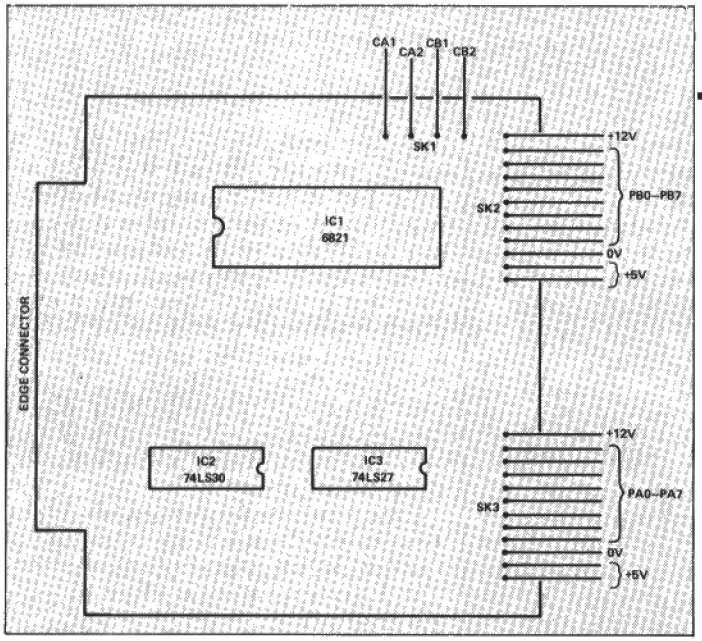


Figure 1. memory address decoding is by 74LS72 and 74LS30 chips.

Figure 2. Overall Circuit Diagram.





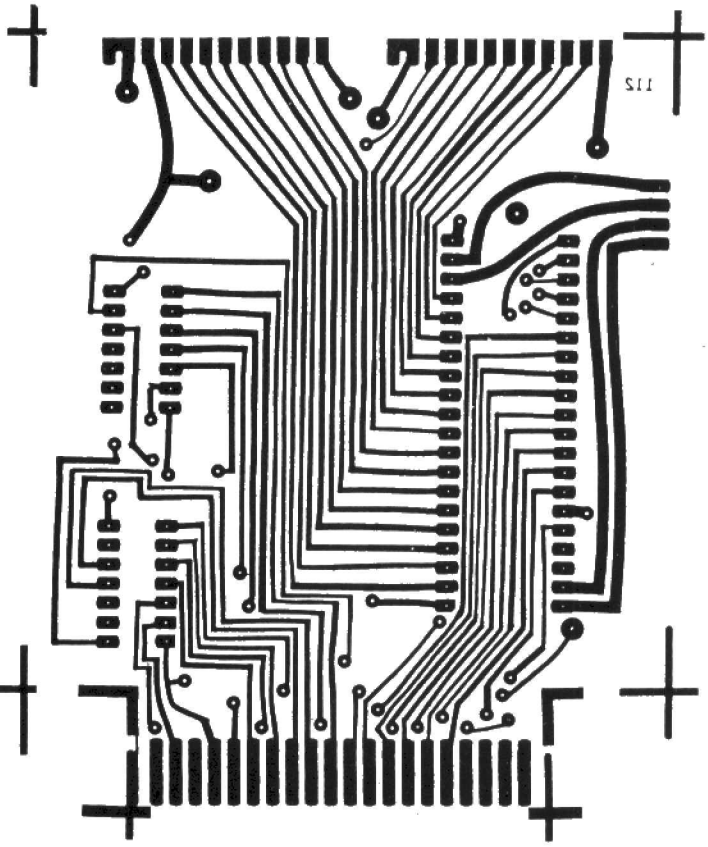


Figure 3 (top left). A simple LED display.
Figure 4 (bottom left). IC's must be inserted as shown.
Figure 5 (above. Foil pattern for underside of PCB.

The full interface and software may be obtained from:
A & C Computers
28 Rowan Way
Lisvane, Cardiff CF4 5TD

LISTING 1.

```
10 REM (C) A&C Computers 1/9/84
20 A1=52960:B1=A1+1:A2=52962:B2=52963:Y1=0
30 A$="SELECT INPUT OR OUTPUT"
40 Z=74:GOSUB 130
50 Y=1
60 Z=1098:GOSUB 250
70 CLS:GOSUB 380:GOSUB 560
80 GOSUB 440
90 Z=330:GOSUB 150
100 Y=0:Z=1098+256:GOSUB250
110 GOSUB 440
120 GOTO80
130 CLS:PRINTA$;:PRINT@64,"PORT A";:PRINT@38,"BIT 7 6 5 4 3 2 1 0";
140 PRINT@134,"BIT 7 6 5 4 3 2 1 0";:PRINT@ 160 ,"PORT B"
150 X=Z
160 K$=INKEY$:IF K$="" THEN 160
170 PRINT@X," ";
180 IF K$="0" THEN  PRINT@X,"0";:X=X+2
190 IF K$="1" THEN PRINT@X,"1";:X=X+2
200 IF X Z+14 AND X Z+25 THEN X=Z+96
210 IF X Z+112 THEN X=Z
220 IF K$="F" THEN RETURN
230 PRINT@X,"*";
240 GOTO 160
250 REM OUT
260 X=128
270 A=0:B=0
280 FOR L=Z TO Z+15 STEP 2
290 IF PEEK(L)=113 THEN A=A+X
300 IF PEEK(L+96)=113 THEN B=B+X
310 X=X/2
320 NEXT L
330 IF Y=1 THEN POKEB1,0:POKEA1,A:POKEB1,255
340 POKEA1,A
350 IF Y=1 THEN POKEB2,0:POKEA2,B:POKEB2,255
360 POKEA2,B
370 RETURN
380 REM IN
390 CLS
400 PRINT@0,"PORTS A&B'S CONDITIONS";
410 PRINT@64,"PORT A";:PRINT@38,"BIT 7 6 5 4 3 2 1 0";
420 PRINT@134,"BIT 7 6 5 4 3 2 1 0";:PRINT@160 ,"PORTB"
430 RETURN
440 A=PEEK(52962)
450 X=128
460 FOR L=1194 TO 1209 STEP 2
470 IF A X-1 THEN A=A-X :POKE L,113 ELSE POKE L,112
480 X=X/2
490 NEXT L
500 A=PEEK(52960)
510 X=128
520 FOR L=1098 TO 1113 STEP 2
530 IF A X-1 THEN A=A-X :POKE L,113 ELSE POKE L,112
540 X=X/2
550 NEXT L:RETURN
560 PRINT@224,"PORTS A&B'S OUTPUTS";
570 E=288
580 PRINT@E+6,"BIT 7 6 5 4 3 2 1 0";
590 PRINT@E+32,"PORT A";
600 PRINT@E+102,"BIT 7 6 5 4 3 2 1 0";
610 PRINT@E+128,"PORT B";
620 RETURN
630 END
```

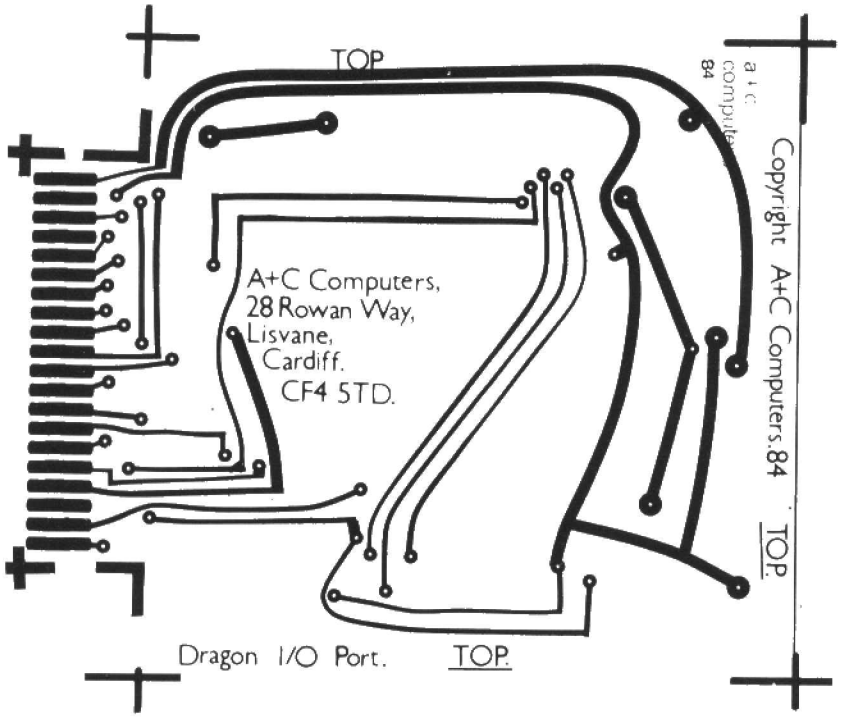


Figure 6. The foil pattern for the top of the PCB.

in **Figure 3** can be used for this. Once again RUN the program and set all the data lines to output – by entering all "1"s – and press "F". Then press the "0" 16 times, which will make both the ports output 0V down the 16 data lines. The 74LS04 will then invert the 0V to +5V and light up all the LEDs. The next stage is to enter all "1"s and press "F" which will make sure all the LEDs will be turned off. It would also be a good idea to enter combinations of 1's and 0's. Once you are sure port A is working, then do the same for port B.

Readers who do not wish to build the interface or enter the program themselves may obtain both, for a total of £20.00 directly, from: *A&C Computers, 28 Rowan Way, Lisvane, Cardiff CF4 5TD.*

PARTS LIST

Semiconductors

IC1	6821
IC2	74LS30
IC3	74LS27

PCB Connectors 0.1in
(Straight polarised locking plug assembly)

SK1	4 WAY
SK2,3	12 WAY

Miscellaneous
PCB; Vero Pins 25; IC Sockets 14,14,40.

Spectrum WAFADRIVE

Has the age of reliable mass storage finally arrived for Spectrum owners? Richard Sargeant looks at Rotronics' Wafadrive and finds favourable results.

The WAFADRIVE is a black-box Spectrum add-on which contains no less than two tape drives, a Centronics standard parallel port and an RS232C serial port and which provides a total of 256K fast-tape storage.

The unit is 230mm broad, 110mm deep and stands 70mm high and looks rather like a small lunch box. It links to the Spectrum via a ribbon cable, and has its own Spectrum port at the back so that your peripherals can still be used. Also along the back surface are two smaller edge connectors, one for the Centronics port and one for the RS232. All these connectors are PCB tongues: the one with the Spectrum signals has the polarity keyway cut in it, but the other two have not. The Wafadrive's makers, Rotronics, say that the printer plugs will be marked clearly as to which way round they should be plugged in. The ones I bought were not, and with a 50-50 chance of getting the connections wrong, I managed it. However, the signals on the connectors ensure that nothing ghastly happens. My Wafadrive and RS232 printer survived the wrong connection, and a quick look at the Centronics specification seems to indicate that a reversal here would also leave the equipment unscathed. Even so, I prefer my connectors to be foolproof.

The unit takes its power from the 9 volt Spectrum supply, and the extra ICs tend to make the Sinclair transformer get a little warm, but there were certainly no problems of overheating. The connections to the Spectrum are tight, and the unit never caused the computer to crash. A red LED indicates that the Wafadrive and Spectrum are switched on, and separate LEDs come on when each drive is in use.

Endless loops

The wafers contain an endless loop of specially-developed magnetic tape which is driven at high speed past the single read/write head. The tape is 1/16th inch wide and has different lengths to provide wafers of 16K, 64K and 128K capacities. Data transfer rate is 2K/sec and access time varies from about 5 seconds on a 16K wafer to a "worst case" time of 45 seconds on a 128K wafer.

Comparison of the industrial BSR waferdrive and the Sinclair microdrive is interesting. At great expense I took the plunge and broke open a 64K wafer and 100K microdrive cartridge and had a close look at the workings of each. The actual drives don't need such brutal treatment – their innards can be viewed with a torch! The mechanics of each cartridge are different in a number of subtle ways, as can be seen by looking at the sketches in **Figure 1.**

Sinclair opted for fast tape speed, which allows short lengths of tape and physically small cartridge size. BSR favoured slower speeds, which means more tape. What is not apparent simply by looking at the hardware is that Sinclair uses a stereo head to read/write two separate channels, while BSR stays with the mono one-channel approach. Therefore, even at identical tape speeds, the BSR system always needs twice the amount of tape to record the same number of bytes. The question now arises as to whether the microdrives are an extremely clever piece of design, or do they in fact suffer from this high-speed approach?

Both systems have a hub (H), a spool of tape (S) and a turntable (T). The path of the tape is shown by the dashed lines, and in both systems the tape turns through 90° as it is pulled out of the inner coil of the spool. The Sinclair system uses two moving guide-wheels, one of which presses against the rubber capstan (C), which is fitted directly to the motor spindle. The BSR system has two static guide grooves (which look like wheels in the diagram) and a rubber idle wheel (I). The BSR capstan is the motor spindle, and is not shown on the diagram. Both cartridges have a head pressure-pad, but the BSR one is more substantial. and a second pressure pad in the BSR cartridge puts the tape under the necessary tension.

The fact of the matter is that there is room to spare in and around the wafer. It has a shutter which closes over the tape whenever the case is removed from the drive and the significance of having the rubber friction wheel in the consumable wafer cannot be under estimated – damage to the Sinclair capstan causes read/write errors on all microdrive cartridges, whereas damage to the BSR idle wheel means the loss of data on just one car-

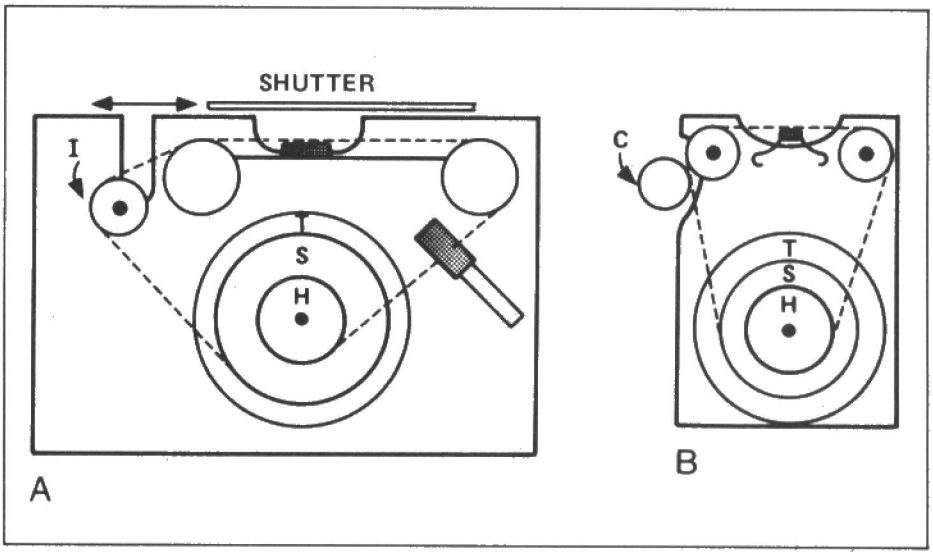


Figure 1. The Wafadrive and microdrive cartridges compared.

tridge.

Bigger is beautiful

BSR physically and electrically swamp their slow-running tape with signal. Sinclair, on the other hand, has to read and write to staggered, separated tracks at high tape speeds. The conclusion is obvious, the BSR system will tolerate wear and damage to the tape surface (particularly the tape edges), the pressure pad and the tape guide system. The Sinclair system is less tolerant of such things.

As a quick aside, it is interesting to speculate on what the microdrives would be like if the "saturate the tape" principle had ever been adopted. There seems to be room in the cartridge for a great deal more tape, so the total capacity of a mono microdrive could still have been around 100K.

Proof of the pudding

When the Wafadrive is first attached to the Spectrum it acts only as a Centronics port. All Spectrum software can be run because the memory map is not tampered with in any way. You could even contrive to fit Interface 1 and a Microdrive or two if you thought that the 9 volt supply could stand up to it! However, if you type NEW* the Wafadrive ROM switches in, the Spectrum memory is reconfigured and you get the Wafadrvie sign-on message on the screen. Just over 2K of RAM has been grabbed by the system (that's rather more than a single Microdrive takes) and so you will find the usual problems of not being able to run software which needed the full 48K, or professional software "protected" by routines which check for an unaltered Spectrum. This happens with the Microdrives fitted, of course, but there you must unplug Interface One to regain normality. In the case of the Wafadrive, you key in NEW and get all your memory back again. This, I thought, is an operating system that I'm going to like.

The user manual is very informative, and is suitable for beginners. You could, for example, buy a Spectrum (or Spectrum Plus) and a Wafadrive at the same time and not have too much trouble getting to grips with the system. CP/M is never mentioned (a good point) and the disc operating system (DOS) and its commands are introduced carefully and with plenty of program examples to illustrate various points. FORMATTING the wafer and reading the directory with CAT are the first commands dealt with, but MOVEing programs to make back-up copies is dealt with later on. Included in the price of the Wafadrive package is a wordprocessor on one wafer, and a blank, unformatted 16K wafer.

The wafer pushes easily into the drive, displacing the drive dust-cover as it does so, and then sits very firmly in place with about ¼" protruding. The lefthand drive is drive a:, the default drive. Formatting my 64K wafer took about three and a half minutes, with on screen status information changing every second or so. The drives have a two speed motor and the first thing that happens is a high-speed search for the metallic "index" which marks the slices in the tape (where recording must be avoided) and the approximate position of the area designated for the directory.

Having found the index, the drive then counts the tape length in sectors and reports back to the directory area. At the slower speed each 1K sector is then formatted and verified. All the while the sectors are being counted off on the screen (my wafer had 71 sectors) and at the end a "Verified good" and "Format completed" signal is given. The 16K wafer (17 sectors) and the 128K wafer (134 sectors) took proportionally shorter, and longer times to be formatted. In use, it seemed that all programs tended to load in under a minute,

"The BSR system will tolerate wear and damage to the tape surface, the pressure pad and the tape guide system . . . the Sinclair system is less tolerant".

```
CAT*"a:"
Display wafer directory for specified drive.
CAT#"a:"
Resets the specified drive for read/writes.
CLEAR*
Closes streams and closes opened data files.
CLOSE#*stream
Closes streams previously OPEN#ed or OPEN#*ed.
CLS*
Clears screen but also resets INK, PAPER, BORDER.
ERASE*"a:filename"
Removes a file. Wildcard facility available.
FORMAT*"R";baudrate
Set RS232 baudrate. 1200 is the default setting.
FORMAT*"a:wafername"
Format and name a wafer.
INKEY$#stream;variable
Read one character from a channel via a stream.
INPUT#stream;variable
Read data from a channel via a stream.
LIST#stream,line
List to a stream, starting at line number given.
LOAD*"a:filename"
Load a program -- accepts BASIC or machine code.
LOAD*
Load first program in directory of default drive.
MERGE*"a:filename"
Merge a BASIC file.
MOVE*"a:filename"TO"b:filename"
The drives may be different or the same.
Wildcards may be used.
NEW
Return to unexpanded BASIC
NEW#
Clear BASIC but stay with Extended BASIC.
OPEN#*stream,"port"
Open a stream to a port.
OPEN#*stream,"a:filename"
Open a stream and assign it to a data file channel.
PRINT#stream;"string",data,variables
Send a string or numeric data to an output stream.
SAVE*"a:filename"LINErun
Save BASIC. LINErun is optional.
SAVE*"a:filename",start,length,run
Save machine code. The auto run is optional.
SAVE#
Overwrite a same-name file.
VERIFY*"a:filename"
Verify BASIC or machine code.
```

The unit, 230mm broad, 110mm deep and 70mm high looks rather like a small lunch box!

with the best times being ten seconds or so obtained from programs held on the short 16K tape. It can take as long as 45 seconds to fetch the directory from a newly inserted 128K wafer, but subsequent directories from the same wafer can be called up in one second, because they are stored in RAM. This neat trick, although taking up some memory space, cuts out a tremendous amount of tape wear and tear, and helps close the gap in operating speed between this system and a disc unit.

The syntax of the new commands is simple. The * is picked up by the Sinclair ROM as a keystroke error and the program flow diverted to the error-handling routines. At this point the Rotronics ROM switches in and investigates the cause of the so-called error. If the syntax typed was that of a Wafadrive command then control stays with the Rotronics ROM and the command is executed. This is the method by which the "Extended BASIC" is created, and it's the same principle as that used for Interface One. The user manual lists all the new system variables (102 bytes of them), and machine-coders will have no difficulty in adding their own commands to E-BASIC. Similarly the workings of the wafadrive and the format of the directory and sectors is also explained, which should ensure that the Wafadrives are used to their full capabilities by Spectrum users. Whether they might also be used by software houses as a medium on which to distribute software is a moot point. The same houses have avoided Microdrive cartridges, and it is usually left to the end-user to transfer, and/or convert the cassette-based program to the new media. In such a situation, the detail written into the Wafadrive user manual is of great significance, and I think that users will appreciate the style and contents of this particular manual.

Spectrum streams

The Wafadrive manual explains the stream handling capabilities of the unexpanded Spectrum and goes on to explain how they can be utilised using E-BASIC. Thus to use an RS232 printer with the Wafadrive would entail choosing an unused stream (eg: number 4) and assigning the RS232 channel ("R") to it; OPEN #4,"R":PRINT #4;"Hello" Perhaps you might like to output your listings to a Centronics printer – there's no need to remove the RS232 printer, simply open another channel: OPEN #5,"C":LIST #5. If you don't feel like changing any of your PRINT or LPRINT commands on existing programs, you can divert their action from the screen and from the Sinclair printer by this merged code:

```
1  CLOSE #2:CLOSE #3:
OPEN #2,"R":OPEN #3,"C"
9999  CLOSE #2:CLOSE #3:
OPEN #2,"S":OPEN #3,"P"
```

The existing streams, opened automatically by the Spectrum ROM are first closed, and the two new printer channels are assigned to the respective streams. Line 9999 restores normality. The baud rate on the RS232 channel defaults to 1200, but other values may be selected. The Centronics port can be used at any time using BASIC or machine code, and it need not necessarily communicate with printers! The RS232 channel, on the other hand, can really only be used via the shadow ROM, since it is the ROM which contains the software UART. Full details are given in the Wafadrive manual.

Conclusion

I found the Wafadrive manual explicit and refreshingly honest. They concede that their RS232 printer lead might need rewiring to work on a particular make of printer, and indeed this was so in the case of my Brother EP44. However, a telephone call to Rotronics sorted out the problem, and added one more printer to their list of popular RS232 devices which don't quite use the RS232 signals in the way "the standard" suggests ought to be the case.

"The user manual is suitable for beginners".

The wafers hold as much data and operate as fast as Rotronics claim. I am therefore inclined to believe that they will last for the claimed 5000 to 10000 passes (at least seven years' usage). I liked the commands of the Wafadrive's Extended BASIC and the neat way in which they have been implemented, and the detailed explanation of their operation given in the manual. The age of cheap mass storage has finally arrived for Spectrum owners. It will be interesting to see if it is appreciated.

Wafadrives are available from ROTRONICS LTD, Santosh House, Marlborough Trading Estate, West Wycombe Road, High Wycombe, Bucks HP11 2LB.

Specifications

DRIVES:

Type	Dual BSR "stringy floppy"
Capacity	128K after formatting
Tape Format	Single track, 1K sectors
Data format	FM coded
Transfer rate	10K baud (2K bytes/sec)
Access time	6.5sec (worst) 16K wafers
	45sec (worst) 128K wafers

WAFERS:

Type	1/16th width infinite loop
	50 feet (128K wafer)
Life	5000 passes minimum
Write protect	Removable tab
Tape protection	Automatic Sliding cover

RS232:

Five wire implementation
RXD, TXD, RTS, CTS and GND
Nine Software selectable
Baud rates 110 – 19200

CENTRONICS:

8-bit data, busy and strobe.

Skywave's Multi-FORTH

Steve Oakey, author of "FORTH for Micros", gets to grips with a version of the FORTH language that allows easy access to multi-tasking.

Multi-FORTH arrives as a ROM and a 176 page manual. Fitting the ROM is easy and on typing "FORTH" the system responds correctly. This package costs £40 plus p&p (£2 in the UK) plus VAT (£6.30), though currently there is a £5 (+ VAT) surcharge because of the high cost and short supply of EPROMs, making a total of £54.05. However, Skywave have just announced the release of Multi-FORTH on disc for use with sideways RAM boards, for a total of £36.80 inclusive.

Skywave have also announced a De-Luxe System, with lots more code, including facilities for the generation of user "windows" which allow several tasks to use different parts of the screen for their output. This De-Luxe System, together with an advanced user manual, costs an extra £40 plus VAT etc – in other words, an extra £48.30. This De-Luxe System was not provided for review.

Multi-FORTH is but a version of FORTH which allows the user easy access to multitasking. In other words, it allows you to get the computer to work on several jobs, or "tasks", apparently simultaneously. FORTH is an interesting language in its own right, using Reverse Polish notation for writing expressions and a stack for evaluating them, as is done with some calculators. Thus instead of writing the Basic statement

```
PRINT 6*(20—7)+5
```

we would write the equivalent FORTH statement

```
6 20 7 — * 5 + .
```

where the full stop is the FORTH equivalent of the BASIC statement PRINT.

This example is rather trivial, and a second might be more useful. Most implementations of FORTH (Multi-FORTH included) come without any array facilities. However, it is easy to include them by writing a simple piece of code, itself written in FORTH:

```
: ARRAY CREATE 2* ALLOT DOES>
        SWAP 2* + ;
```

and we now have one-dimensional array facilities for integers, like those of Basic. Thus we can declare a 20-element array named TOTALS by writing

```
20 ARRAY TOTALS
```

and can store the value —5 in element number 16 by writing

```
—5 16 TOTALS !
```

as the equivalent of the BASIC statement

```
TOTALS%(16) = —5
```

because the "!" is the FORTH assignment operator, the equivalent of the BASIC "=".

The popularity of FORTH derives from two features. First, FORTH programs execute extremely quickly – very much faster than BASIC (it is usual to quote a factor of ten difference), and fairly close to the speed of compiled programs. At the same time, FORTH is very much easier to write and debug than programs written in assembler, which is the other main method of obtaining speed of execution. Thus you can have all of the advantages of machine code speeds without the disadvantages. Having said that, I must say that simple counting loops took approximately as long to run in FORTH as they did in Basic.

FORTH comes complete with an assembler built in to provide extra speed of execution though that should rarely be needed.

". . . allows the computer to work on several tasks . . ."

The second useful feature of Forth is one that applies to very few languages indeed. You can define new features of the language itself. As a simple example, if you prefer to stick to the word PRINT rather than a full stop for your "print" statement, you can do just that by writing (in FORTH)

```
: PRINT . ;
```

We could now write the previous example as

```
6 20 7 — * 5 + PRINT
```

FORTH is particularly useful for real-time control, combining flexibility with speed of processing and ease of programming. But it is also particularly good for advanced games, where speed is paramount. As an example of a game written in FORTH, you might like to look at the winner of the BYTE Game Contest, published in BYTE, December 1982, pages 124-138.

Multitasking is remarkably easy to do with Multi-FORTH. As a simple example, I defined a piece of code to count upwards from 1, printing the sequence of numbers on the screen as it went. I called it COUNT, using the standard FORTH definition:

```
: COUNT 0 BEGIN 1+ DUP . AGAIN ;
```

to provide an infinite loop. I then started up two separate tasks to do this same count and print job by merely typing

```
RUN COUNT
RUN COUNT
```

and the two tasks (which in this case did the same job as each other) ran concurrently.

However, this highlighted a general problem with multitasking: the use of shared resources. In this case, both tasks wanted to output to the screen, and the output was not only jumbled, but in some characters were actually lost. This problem can be readily overcome by using semaphores, which Multi-FORTH does provide. The use of semaphores allows tasks to share resources in an intelligent way. I defined a semaphore called SCREEN.IN.USE by writing

```
1 SEMAPHORE SCREEN.IN.USE
```

where the "1" indicates how many tasks are allowed simultaneous access to the resource. I then defined a new print function PRINT to be

```
: PRINT SCREEN.IN.USE WAIT .
        SCREEN.IN.USE SIGNAL ;
```

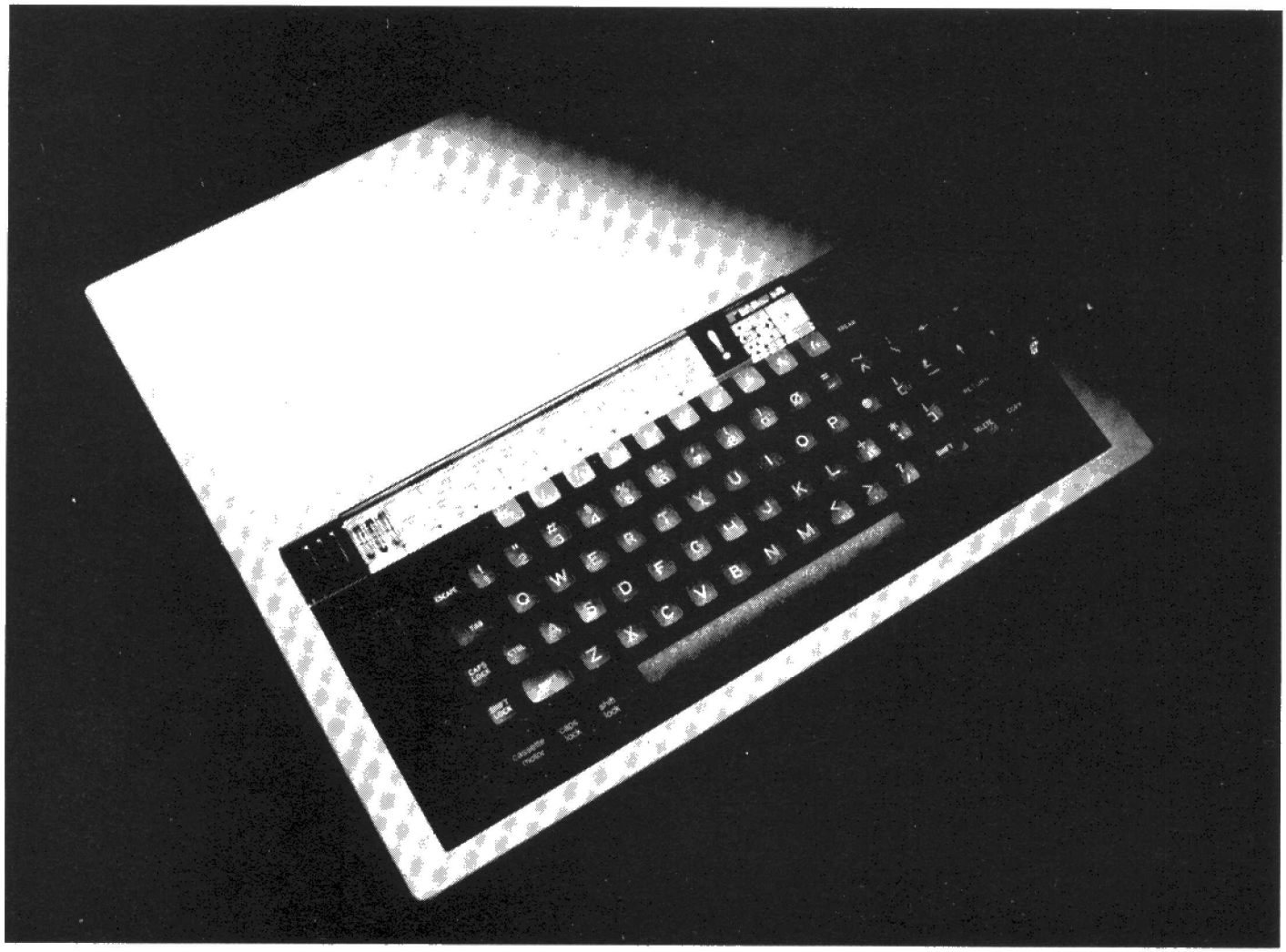
and then redefined COUNT as

```
: COUNT 0 BEGIN 1+ DUP PRINT
AGAIN ;
```

and the two tasks ran as I wanted: the output was mixed together, but no characters were lost and digits from each of the two tasks were not intermingled.

WAIT is used to hold up the calling task if more tasks want the resource than are allowed it. SIGNAL is used to indicate that the calling task has finished with the resource. All the programmer needs to do is define a semaphore name associated by the programmer with the resource that is to be shared. By using the semaphore differently, I could force one task to wait until the other task had output all of its values to the screen, avoiding any mixing together of output. This would obviously be useful for several tasks sharing the printer, where you would almost certainly not want the separate outputs to be intermingled. There are also facilities for accessing the variables of another task, allowing one task to monitor the actions of another. This allows you to write very advanced programs, including complex control software.

The only unfortunate thing about all of this is that the manual fails to give an expla-



nation of these facilities.

Skywave claim that the MultiFORTH software is based on a completely new version of the FORTH '83 Standard. It is worth pointing out that while this claim is true, most of the "new" 83 definitions are trivial changes from the '79 Standard, and the manual is at times unsure of whether a definition is the same as the '79 Standard or not. Unfortunately, the FORTH '83 Standard does have apparently minor – but significant – modifications to some routines. For example, the equivalent of the BASIC DIV routine now rounds downward, whereas in the '79 Standard it rounds toward zero. This will be of considerable effect to those who want to run software written assuming the '79 Standard.

What about real problems with the software? FORTH itself carries out few checks on its evaluation process. Thus if you try to multiply two numbers when only one is on the stack, you will not get any indication of an error. Multi-FORTH is no exception to this rule, and this creates a real problem when using the built-in editor. FORTH source code is usually stored in "screens" on backing store, whether on tape or disc. To insert or change a program, you need to type the screen number followed by the word EDIT. If you forget the screen number, you may well wipe the whole file!! (Yes, I managed to do just that).

I do not like the editor, but it is not an unusual FORTH type. It allows use of the screen copying key with cursor controls, but those used to proper screen editing will find a considerable difference.

Other oddities include the following: if you input a new version of a line and actually input zero characters, the original line is unchanged, but no warning of this is given; the comment character is "(" rather than BASIC's "REM", and this for no obvious reason gives an error if in column 1 of a line; one screen has spurious, non-printing character in it, which gives an error message and aborts the compilation.

"... a good introduction to the language ... very useful for experimentation ..."

Another "feature" of Multi-FORTH that may be unacceptable is the time taken to output numbers to the screen. Output is usually the slowest part of any piece of software, but there seems to be no obvious reason for it to take more than twice as long as BASIC!

It must be said that the FORTH system has been nicely adapted to the BBC computer. Access to the operating system is provided in the usual way, though the parameters are written in a different way to fit in with FORTH's Reverse Polish notation. Thus to select mode 5 with foreground colour red and background colour yellow, you would write

```
5   MODE
1   COLOUR
131 COLOUR
```

Similarly, other (non-OS) commands would have the command following the parameters.

Multi-FORTH also allows OS commands to be issued from within it and from within programs, though the latter facility needs a minor "fiddle" for it to work. As the manual explains, if you write a piece of FORTH code that wishes to issue a *SPOOL command, the command would have to be followed by a ^ character – a very minor addition. The only real difficulty you might find is that the plot command is more limited than the BASIC version, simply because FORTH does not have as standard a floating-point package.

Multi FORTH is significantly faster than BBC BASIC, at least in its arithmetic facilities, but much slower than BASIC when printing numbers to the screen. Unless you are interested in real-time systems, possibly for control purposes, I doubt that you would want to buy it for practical applications. Having said that, it is an interesting pice of software to play with, a good introduction to the language, and very useful for the purposes of experimentation.

Parlez Pascal

Gerry Davies concludes his Pascal tutorial series with a look at the features that make Pascal a structured language.

In this, the last part of this series, we look at some of the remaining features of Pascal. In particular, we see the type that makes Pascal a structured language, the RECORD. We will also have a brief look at sub-programs which allow you to split your program into sensible sized chunks that are easy to write and test. It is this ability to structure your program that makes Pascal suited to solving large problems.

Logical connectives

So far, we have said little about BOOLEAN variables. It is now time to briefly look at the operations you can perform on them. As they represent logical values, the standard logical operations of Boolean algebra are available, namely AND, OR and NOT (operations such as NAND and EXOR can be built up from these three basic operations so are not included). It is important to realise that these operations can only be performed on BOOLEAN variables, or BOOLEAN expressions. You cannot use AND to perform a bit-wise AND operation on an INTEGER.

These operations perform the usual functions, so 'A AND B' is true only if both A and B are true. Similarly, 'A OR B' is true if either A or B or both are true. Also 'NOT A' is true when A is false, and vice versa. These connectives are commonly used in IF statements, such as:

```
IF (Today = Wed) AND (Time > 20)
   THEN . . .
```

Notice that you must use brackets around the values to be ANDed. This is because AND has a higher precedence than comparison, so if they were not included we would attempt to compute 'Wed AND Time'. As these values are not BOOLEAN, an error would occur.

Operations on enumerated types

Although we mentioned enumerated types in the last part, nothing was said about the operations that can be performed on them, of which there are only two. You cannot perform standard arithmetic on an enumerated type. This is because it has no meaning, after all what does '2 * Tuesday' mean? Similarly, you cannot add numbers to them, which may seem a bit odd, after all you might expect to be able to add 1 to 'Tuesday' and get 'Wednesday'. But what do you expect if you declare

```
TYPE
Gender = (Male, Female) ;
```

What does 'Male + 1' mean now?

As operations such as finding the next value of an enumerated type are needed, Pascal provides a mechanism. To find the successor to a value, the next possible value, you use SUCC. So given

```
TYPE
   Days = (Sat,Sun,Mon,Tue,Wed,Thu,
   Fri) ;
VAR
   Day : Days ;
```

SUCC(Day) will find its successor. So

```
Day :=Tue ;
Tomorrow :=SUCC(Day) ;
```

results in 'Tomorrow' having the value 'Wed'. Similarly, PRED finds the predecessor of a value. Given the above values,

```
Yesterday :=PRED(Day) ;
```

would assign the value 'Mon' to the variable 'Yesterday'.

"... sub-programs make Pascal suited to solving large programs ..."

The important point to notice is that PRED and SUCC are not circular. Therefore SUCC(Fri) does not give 'Sat', it gives an error message! Indeed many enumerated types are not circular in this way; for example the type 'Gender' declared above. However, it is straightforward to implement a circular operation when we need one, by testing to see if the variable has already got the last possible value. In the case of variables of type 'Days', we must test to see if the variable is already 'Fri'. If it is, we can assign the value 'Sat', otherwise we can simply use SUCC in the normal way:

```
IF DAY = Fri
   THEN Day :=Sat
   ELSE Day :=SUCC(Day) ;
```

Despite the sparse range of operations that can be performed on enumerated types, they are still very useful. Don't forget that you can use them also in 'FOR' loops and as ARRAY indicies as previously seen. Once again, remember that using suitable names for your enumerated types can make your program very easy to read. It is worth choosing names so that your program reads as much like English as possible – then you don't need anywhere near as many comments! This technique is known as writing self-documenting code.

The CASE statement

The last control statement in Pascal is the CASE statement. The 'IF' statement allows you to select between two courses of action based on the value of a BOOLEAN variable, or a relational comparison yielding a BOOLEAN result. The CASE statement allows a multi-way selection of actions dependent on the value of a single variable, which need not be BOOLEAN. Any CASE statement could be implemented with a block of nested IFs but this would not be as clear or efficient.

A good example of its use is with enumerated types. We have just seen that there are only two operations defined for enumerated types, PRED and SUCC. In particular, you cannot WRITE such types and get the expected result. Given the above declaration of the type 'Days', WRITELN(Sat) would give you the value of the internal representation of 'Sat' (probably a zero). If you actually want to write out the value of 'Day' in English, you can use a CASE statement. Let us suppose that we know 'Day' has a value between 'Mon' and 'Fri' at a given point in our program, then we can have:

```
CASE Day OF
   Mon : WRITELN('Monday') ;
   Tue : WRITELN('Tuesday') ;
   Wed : WRITELN('Wednesday') ;
   Thu : WRITELN('Thursday') ;
   Fri : WRITELN('Friday') ;
END ;
```

This means 'examine the value of Day, and perform the statement associated with the label of the same value'. These selected statements can be anything, including compound statements, or even another CASE statement. An important point to remember is that once the selected statement has been executed, the entire CASE statement finishes (C programmers beware!). Also, if the control variable has a value for which there is no

corresponding label then an error is signalled. You must therefore either provide an action statement and label for each possible value, or ensure that other values can never occur. To help with the former, you can have more than one label associated with each action statement, and you can also have null statements. The above CASE could therefore be made totally secure by including a

```
Sat, Sun : ;
```

within the CASE . . END bracket.

Alternatively, a simple IF will suffice:

```
IF (Day>=Mon) AND (Day>=Fri)
  THEN
    CASE etc
```

Some versions of Pascal allow an OTHERWISE clause in a CASE statement to trap the values for which a label is not found. As this is not standard, you should not rely on it.

To complete the comparison with the IF statement, we will consider how to implement an IF with a CASE. As BOOLEAN variables are considered to be enumerated types, we can use them as the CASE selector. Thus

```
IF A > B
  THEN Max :=A
  ELSE Max :=B ;
```

is equivalent to:

```
Selector :=A > B ;
CASE Selector OF
  TRUE : Max :=A ;
  FALSE : Max :=B ;
END ;
```

SETs

This is a data type par excellence. None of the other common languages provide a data type for representing SETs, which is a great loss for the other languages. The SET type is the same as the traditional set from modern mathematics. Variables of this type have a range of possible values defined when they are declared. Unlike other variables, though, they can take more than one value at any time, or even no value at all!

A typical SET declaration might be:

```
TYPE
  Capitals = SET OF 'A' . . 'Z' ;
VAR
  Letters : Capitals ;
```

This declares 'Letters' to be a variable of type 'Capitals', in the normal way. 'Letters' has no initial value, but can have any of the list of values mentioned in its declaration. In this case, 'Letters' can have any capital letter as its value, or even all of them. The standard rules of set theory apply to Pascal SETs. Thus [ ] represents an empty set, ie a variable with no values assigned to it. When values are assigned to a SET, they must be enclosed in square brackets. Thus:

```
Letters:=['A','B','C'] ;
```

assigned the three letters to the variable 'Letters'. We can add further values to our SET with the union operation, represented by a plus sign in Pascal. The union of two sets is a third set containing all the elements in either set, but with no duplication.

```
Letters :=Letters+['C','D'] ;
```

means find the union of 'Letters' and the two new letters. Thus 'Letters' will now have the value ['A','B','C','D']. Note that there must be no duplication, so the union has but one 'C'. We can also find the intersection of two SETs, that is a SET containing only those elements that appear in both of the intersected SETs. The intersection operation is denoted by a minus sign, so:

```
Letters :=['A','B','C']-['C','D'] ;
```

leaves 'Letters' with a value ['C'].

You can assign SETs in the normal way, and test to see if two SETs are equal. There is also a far more useful operation. While testing to see if two SETs are the same is useful in some cases, it is wise to check to see if each SET has precisely the same number of elements, and the same elements individually. It is often useful just to see if a SET contains a given value, IN does just that.

Suppose we want to ask the user of a program a question to which the answer is yes or no. We could just read in the reply, and compare it with 'YES' and so on. However, it is only necessary to read in the first letter and check if it is a Y or not. To ensure the user can answer in upper or lower case, we must also check to see if the letter is a lower case Y. This can be easily done with SETs:

```
Answer : BOOLEAN ;
Ans : CHAR ;

Answer :=Ans IN ['Y','y'] ;
```

Given the declarations, 'Answer' takes the value TRUE if the answer was a 'Y' or a 'y'.

The full power of the SET becomes apparent when you start writing compilers and operating systems. However, this example shows that it is still very useful in less ambitious programs. One note of caution, though. Not all Pascal compilers will allow you to have a SET OF CHAR as they put an upper limit on the maximum number of elements a SET can have. Check your implementation before you start writing programs (or buy it, if possible!).

"... the RECORD data structure helps set Pascal apart from other micro languages such as BASIC ..."

The RECORD data structure

This is the classic data structure that helps to set languages like Pascal apart from BASIC and the other micro-languages. There is no equivalent in BASIC, though modern languages such as C, ADA, Modula-2 and so on all have a RECORD of some form.

So what is a RECORD? It's just what it sounds like, a group of other data types that can be manipulated as one.

It can be compared with the ARRAY – a data structure you should already be familiar with. In an ARRAY we have a group of data elements, each of which can be manipulated individually or the whole can be referred to. However, each element in an ARRAY must be of the same type which may become inconvenient. For example, suppose we want to store the date in a variable which requires three separate parts – one each for the day, the month and the year. Suitable Pascal declarations of types are:

```
TYPE
  Days=(Mon,Tue,Wed,Thu,Fri,Sat,
    Sun) ;
Months=(Jan,Feb,Mar,Apr,May,Jun,
  Jul,Aug,Sep,Oct,Nov,Dec) ;
  Years=1900 . . 2000 ;
```

Notice we have restricted the range of valid years using a sub-range declaration. We can now declare three variables for each part of a date, and use these in the normal way. However, since all three parts form a whole, we could also declare a RECORD structure to hold a complete data:

```
Date=RECORD
  Day : Days ;
  Month : Months ;
  Year : Years ;
END ;
```

This creates a type called 'Date' that has three constituent parts. We can now declare variables to be of this type in the usual way:

```
VAR
  Today, Tomorrow : Date ;
```

Each of these variables now has three parts within them. You can assign variables, so long as they have the same type, so

```
Tomorrow := Today ;
```

is perfectly valid. You can also access the individual fields (or parts) within the variables. Thus 'Today.Month' refers to the month field of the variable 'Today'. This field of 'Today' has the type 'Months', not 'Date'. We can therefore manipulate it as if it were a normal variable of type 'Months':

```
Today.Month := Feb ;
Today.Month := Tomorrow.Month ;
```

and so on. It is important to understand that 'Today.Month' refers to a sub-part of 'Today', so is distinct from 'Tomorrow.Month'.

As RECORDs are just a further user-defined data type, you can use them wherever a data type is appropriate. You can have ARRAYs of RECORDs, RECORDs of RECORDs, but obviously not subranges of RECORDs. We can design a simple data structure to act as a personnel record which makes use of these features.

Each personnel RECORD will need to hold the employee's name, age, date of birth and salary. We already have a type suitable for the date field, and the age can simply be an INTEGER. The name needs two parts, the first name and the surname (this can clearly be extended at a later date). A suitable type for the name is an ARRAY of characters. We therefore have:

```
Personnel = RECORD
  Birth : Date ;
  Age : INTEGER ;
  FirstName, SurName : ARRAY 1 . . 20
  OF CHAR ;
END ;
```

Notice that the 'Birth' field is a RECORD itself. We can refer to this field as before, if 'Person' is of type 'Personnel':

```
Person.Birth := Today ;
```

We can also refer to sub-fields of 'Person'. So 'Person.Birth.Day' refers to the 'Day' field of the date of birth of the given persons RECORD.

Clearly, the RECORD is a very powerful data structure which can be used to produce very clear programs, so long as sensible names are used for its parts. Its principle value is that any structure can be built, as RECORDs may contain sub-records and so on. Part of the skill in Pascal programming comes from designing suitable data structures for representing the data in your program. Indeed, the correct choice of structure can make some tricky looking problems very easy to solve. However, designing data structures is also an acquired skill. More books have been written on data structure design than on Pascal programming, so you can't expect to pick it up overnight!

Sub-programs

Sub-programs are literally small programs. They have the same format as a normal program, can contain type and variable declarations and can pass values back to the main program. There are two types of sub-program allowed in Pascal, the PROCEDURE and the FUNCTION. The PROCEDURE is similar to a subroutine in other languages, and the FUNCTION also appears in BASIC and so on. The important difference is that a FUNCTION will always return a value, but a PROCEDURE need not.

When a sub-program is declared, it is given a name. It is then called from the main program by using the name, there is no equivalent of a CALL or GOSUB in Pascal. After the name of the routine a list of parameters must appear. When the routine is called, the program will provide a list of values to be passed in to the called sub-program. Thus;

```
Print ( 20 ) ;
```

is a call to a PROCEDURE called 'Print' with a single parameter.

Sub-programs must be declared after the VAR declaration section in a program. 'Print' might have the declaration

```
PROCEDURE Print (I : INTEGER) ;
BEGIN
  WRITELN (I) ;
END ;
```

This declares 'Print' to have a single INTEGER parameter called 'I'. This name will then refer to the value passed when the PROCEDURE is called. Thus in the above example, I will have the value 20 when it is called. The name of the parameter or parameters is local to the sub-program. It can therefore be the same as a variable name in the main program, but will not refer to the same variable.

"... RECORD allows very clear programs to be produced ..."

A FUNCTION declaration is similar, but we also must define the type of the value the FUNCTION is to return. So

```
FUNCTION Max (A, B : INTEGER) :
INTEGER :
BEGIN
  IF A > B
  THEN Max := A
  ELSE Max := B ;
END ;
```

declares a FUNCTION with two INTEGER parameters that will return an INTEGER result. The two parameters, A and B are compared in the FUNCTION body, and the larger of the two is returned by assigning it to a variable of the same name as the FUNCTION. This notional variable is declared automatically and this FUNCTION can be used anywhere in your program, so

```
Largest := Max(X, Y) ;
```

will assign the value of the larger of the two variables to 'Largest'.

Sub-program bodies can contain as many statements as you wish, all enclosed in the traditional BEGIN . . END bracket. In particular you can declare further types or even sub-programs. These declarations fit in between the name declaration and the first BEGIN, just like declarations in the main program. All variables, sub-programs and types declared within a sub-program are local and cannot be used by the main program. However, all declarations in the main program are considered to be global, so can be accessed within the sub-program.

One type of parameter that is of interest is the VAR parameter. Those parameters we have seen so far are implemented by copying the value used in the calling routine into the variable named in the sub-program. This in the above FUNCTION, the values in 'X' and 'Y' when 'Max' is called are copied into 'A' and 'B'. The values are not copied back afterwards, so if they are altered in the sub-program, they are not altered in the main program. Often we want to alter a value and return it to the calling program. This is what VAR parameters are for. If a parameter is declared as a VAR parameter, then when the sub-program is called, an address of the variable is passed, and the sub-program manipulates the variable whose address it is given. Thus all alterations of values within the sub-program are reflected in the calling program.

We can use VAR parameters to implement the 'Max' operation with a PRO-

CEDURE. Thus:

```
PROCEDURE Max1 (A, B : INTEGER ;
VAR Largest : INTEGER ) ;
BEGIN
  IF A > B
    THEN Largest :=A
    ELSE Largest :=B ;
END ;
```

this would be called with

```
Max1(X,Y,Largest) ;
```

to have the same effect as the FUNCTION.

The use of FUNCTIONs and PROCEDUREs deserves a series of articles of its own, however we have seen enough to get the general idea.

String Handling

This is one of the weak areas in Pascal. While strings can be handled, BASIC provides more useful operations on strings. As we saw in the section RECORDs, strings can be held in ARRAYs of characters. The problem with this method is that all strings must be of exactly the correct length to fit in the ARRAY. Thus given

```
TYPE
  String=ARRAY [1 . . 20] OF CHAR ;
```

strings of this type must all contain exactly 20 characters. Thus assignments to these strings can be rather cumbersome, as the remaining characters must be filled with spaces. If 'Name' is a variable of type 'String', then

```
Name :='Fred
```

is a typical assignment! Whilst this is useable, it can be rather inconvenient. Some Pascals therefore provide a pre-defined type for holding strings. This is normally a RECORD of the form:

```
String=RECORD
  Words : ARRAY [1 . . MaxLength] OF
  CHAR ;
  Length : 0 . . MaxLength ;
END ;
```

where Maxlength is a predefined constant. Therefore, the 'Length' field is used to tell you how many of the characters in the ARRAY are currently valid. It is a worthwhile exercise devising a suitable set of routines to use such a type if your compiler does not provide one.

FILEs

Files are a rather thorny subject in Pascal. Many people feel that the treatment of files in the language is less than satisfactory. Whilst this is true in general, you must remember that Pascal was only designed as a teaching language, not a full production language. Various people have attempted to add different features to Pascal to extend its file handling capabilities, so I shall restrict myself to just the common ground. When you come to use Pascal on your system, consult the user guide for your compiler to check on the details for your implementation.

Files consist of a collection of records. However, we shall call them elements to save confusion with the RECORD data structure. All elements in a file must be of the same type, which can be any of the Pascal types or user defined types, including RECORDs. Whenever you READ or WRITE to a file, you must transfer an entire element. Let us look at a file declaration:

```
TYPE
  Number=FILE OR REAL ;
VAR
  Numbers : Number ;
  Value : REAL ;
```

"... you should now be able to read Pascal programs ... reading well written programs is the best way to learn".

This declares a file of REALs, with a file variable named 'Numbers'. Before we can use this file, it must be opened. If you wish to WRITE to the file (open it for output) you must open it with REWRITE. If you want to READ from it, you must use RESET. Thus

```
RESET (Numbers) ;
```

will open our file for reading. Notice we have said nothing about linking our file to a real file in your computer. This is one of the undefined areas of Pascal mentioned above! In general, if you want your file to exist after running your program, you must include its name in the PROGRAM statement, but you must look at the documentation of your compiler.

Having opened the file you can READ from it in the same way as you would read from the keyboard. However, you must say what file you are reading from. So

```
READ(Numbers, Value) ;
```

reads from the file 'Numbers' a single element into the variable 'Value'. As with files on all systems, the data is held sequentially. Once you have read an element from a file, there is no way of going back and reading the same element again, without starting from the front of the file again. Some extensions allow direct access to individual elements in a file, but this is non-standard.

The INPUT and OUTPUT files listed in the PROGRAM statement are predeclared files of type TEXT. With files of this type, you can use READLN and WRITELN in the usual way. These procedures will not work with other files. You can declare your own files of type TEXT in the usual way, remember that TEXT is a predefined TYPE, so you only need a VAR declaration:

```
VAR
  Book : TEXT ;
```

declares a file of type TEXT.

There are two useful FUNCTIONs associated with files. The first is EOF. This returns TRUE if the end of a file is reached. Thus

```
WHILE NOT EOF(Book) D0
BEGIN
  . . . . . .
```

will work through the file 'Book' until the end of the file is reached. Similarly, EOLN will return TRUE if the file is a TEXT file and the end of a line has been reached.

We can use these two FUNCTIONs to copy the file to the screen. Thus the following section of code will write the contents of the file 'Book' to the OUTPUT file of your program, the screen:

```
WHILE NOT EOF(Book) D0
BEGIN {Read a line}
  WHILE NOT EOLN(Book) D0
  BEGIN
    READ(Book, Ch) ;
    WRITE(Ch) ;
  END ;
  READLN(Book) ;
  WRITELN ;
END ;
```

Notice that the inner loop reads in a single letter from the 'Book' file and writes it out to the screen. This is repeated until the end of line is encountered. Then the loop drops through, and the program reads the end of line character with a READLN and sends a carriage return to the screen with a WRITELN. This continues until the entire file has been copied. ('Ch' is obviously declared as a variable of type 'CHAR'.)

Whilst these few operations are sufficient for basic file handling, there is obviously scope for improvement. However, files tend to be a rather grey area in all languages, so perhaps we shouldn't complain too much.

Conclusions

Hopefully this all to brief look at Pascal has given you a flavour of the language. You should now be able to write small programs yourself. More importantly, you should be able to read Pascal programs. Examining well written programs is an excellent way to learn a language, so I suggest you try to find some Pascal to read. These will also introduce you to the few features we have not covered in these articles, namely pointers and varient records. While these features are very powerful and useful, they are beyond the scope of an introduction. It is not until you have written a fair number of Pascal programs that you would appreciate the need and beauty of linked lists built up with pointers.

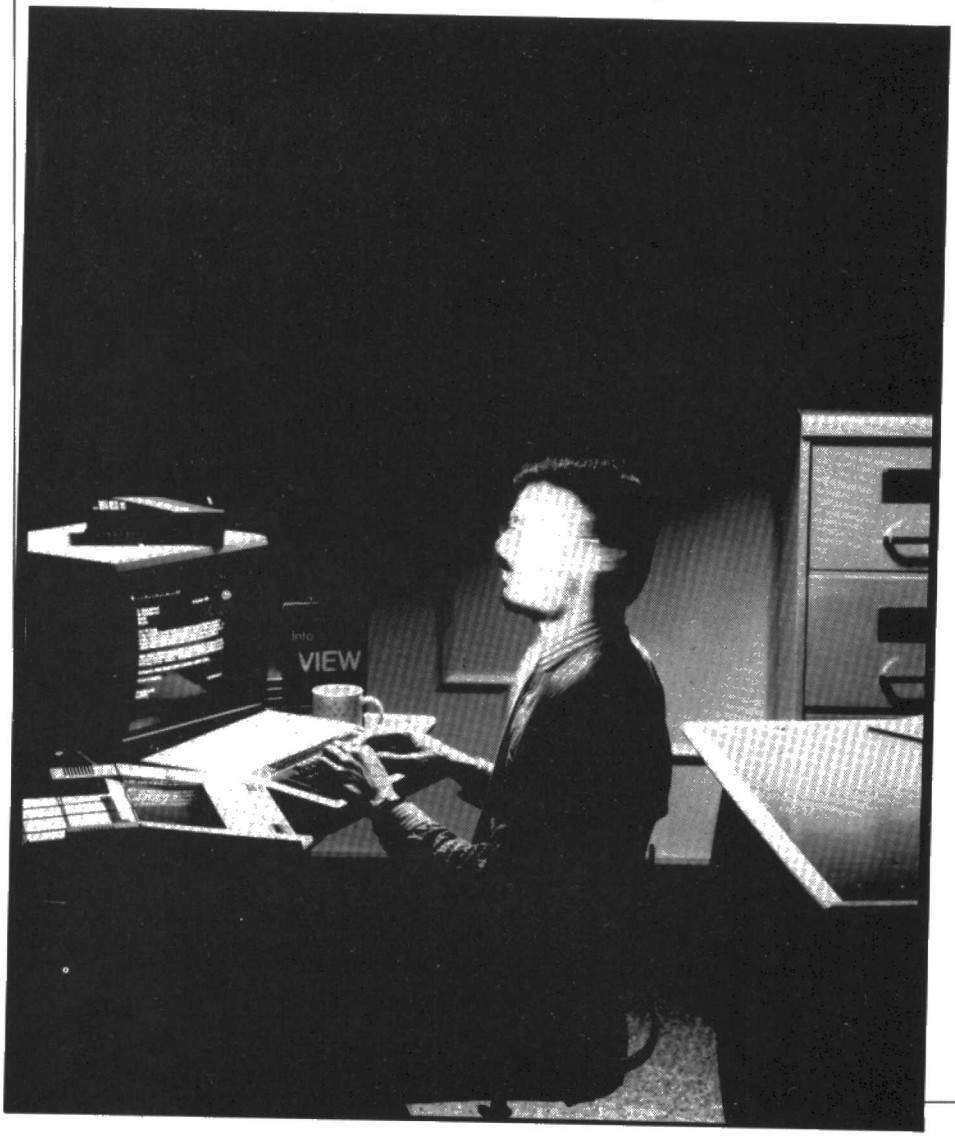
However, there are a number of books which will teach you more about Pascal. One that I would strongly recommend is 'Software Tools in Pascal' by Kernighan and Plauger, published by Addison-Wesley. This takes you through the design of a number of tools, such as text processors and editors. It will teach you good programming practices, and give you some examples to learn from.

BBC PRINTER BUFFER

Brian Alderwick and Peter Simpson's latest project allows any single bank of sideways RAM to function as an extended printer buffer.

Most printers in use with microcomputers have very small built-in buffers; in addition the BBC micro has only a 64 character printer buffer: this means that the micro is often limited in speed by the rate at which the printer functions, and the micro is tied up longer than necessary during printing operations.

This article describes a piece of sideways ROM software which will enable any single bank of sideways RAM – ie up to 16K – to function as an extended printer buffer. The buffer operates identically to the original BBC buffer which it completely replaces and functions with both serial and parallel printers and user defined printer drivers. Five new star commands are provided to control the buffer and comprehensive *HELP messages are included. The extended buffer uses no private or public workspace during execution and is completely transparent to other software. The buffer will retain its state across a CTRL/BREAK although the contents will be lost.



In the BBC micro the printer buffer size is 64 bytes (characters). In addition most printers have a one line 80 or 132 character buffer. Thus, since the printer is relatively slow, about 80 to 160 characters per second (cps) for a dot matrix device and 10 to 50 cps for a daisy wheel device, and the buffers are relatively small, the computer will only be able to print characters at the speed of the printer and will become available for other work just one line before the printer finishes.

Imagine now that the buffer size was 16K bytes. If the text to be printed was less than this, then the micro would race away at its maximum speed storing bytes in the buffer until it reached the end of its task. The buffer would then continue to dispense text to the printer when required, while the micro could be given another task. If the text to be printed was greater than 16K bytes long then the first 16K of text would be delivered to the buffer by the micro at full speed but it would then have to wait, inserting characters into the buffer at the rate at which the printer removed them. Ultimately it would finish its task and the remaining 16K of text would be printed by the buffer whilst the micro was free to do other work. Thus the buffer is still performing a useful task by allowing the micro to do other work whilst the last 16K of text is printed. A universal rule about buffers is that the bigger they are the better!

Use of the buffer

When first powered up the printer buffer will be switched off and the micro will perform completely normally using its standard 64 byte printer buffer. The buffer is activated by the *BUFFON command. This checks that some sideways RAM is present. If no RAM is found an error message is given. A further check is then made to ensure that the existing printer buffer is empty, if not then an error message is given and the command is aborted. If RAM is found then its ROM slot number is compared with that of the current language. If these are identical the command will be aborted since you would not want to risk corrupting the current language. If the

"... A universal rule about buffers ... the bigger, the better ..."

RAM does not contain the current language then the operating system is interrogated to find out whether it contains a valid sideways ROM image. If not then initialisation proceeds, however, if a ROM image is found then a request to continue is issued. Finally, a message about escape key effects is printed. At this point the buffer is fully activated and behaves identically to the original BBC buffer except that it has a capacity equal to the amount of sideways RAM fitted. The *BUFFOFF command de-activates the buffer but only functions when the buffer is empty. The

*PURGE command clears the buffer as do *FX21,3 and *FX15,0.

Two additional commands are provided namely *PURGEON and *PURGEOFF. These determine whether OSBYTE 126 (*FX 126), the ESCAPE key acknowledge routine, will purge the printer buffer or not. The default setting is not to clear the buffer and this is the only difference to the performance of the built-in BBC buffer which is cleared by this call. This departure from the norm has been made for the benefit of users of VIEW. This word processor uses the ESCAPE key to toggle between command mode and the text mode.

"Five new star commands are provided to control the buffer and comprehensive *HELP messages are included".

The situation could easily arise wherein a large piece of text is dumped into the buffer and a second piece of text loaded ready for examination. Use of the ESCAPE key to toggle into text mode would then, under normal circumstances, clear the printer buffer before the text contained in it was printed. Hence the behaviour of the buffer has been modified to prevent this occurring. Execution of the *PURGEON command will reverse this situation and allow the buffer to be cleared by the ESCAPE key. These commands will only operate if the buffer is first activated by *BUFFON.

The controlling software is intelligent enough to realise when it itself is in sideways RAM, and if this is the case, then the buffer start address will be set to a value which is just above the finish of the controlling software thus avoiding corruption of the controlling code during operation. The software is just over 2.5K long. If the *FX5 call (select printer destination) is issued it waits until the printer buffer is empty, as is proper, before executing so the benefit of a large buffer will be lost if this call is issued whilst the buffer is not empty.

Principles of buffer operation

This buffer is of the FIO type (First-In-First-Out). As the name implies the first character to be removed from the buffer will be the first character that was inserted and so on. The buffer is controlled through four pointers and two flags. Two of the pointers contain the addresses of the start and finish of the sideways RAM memory in use as the buffer. They are determined when the buffer is activated and do not change during operation. In the software they are called <bs> for buffer start and <bf> for buffer finish. The other two pointers contain the addresses of the next character to be removed from the buffer and of the next empty space in the buffer which could

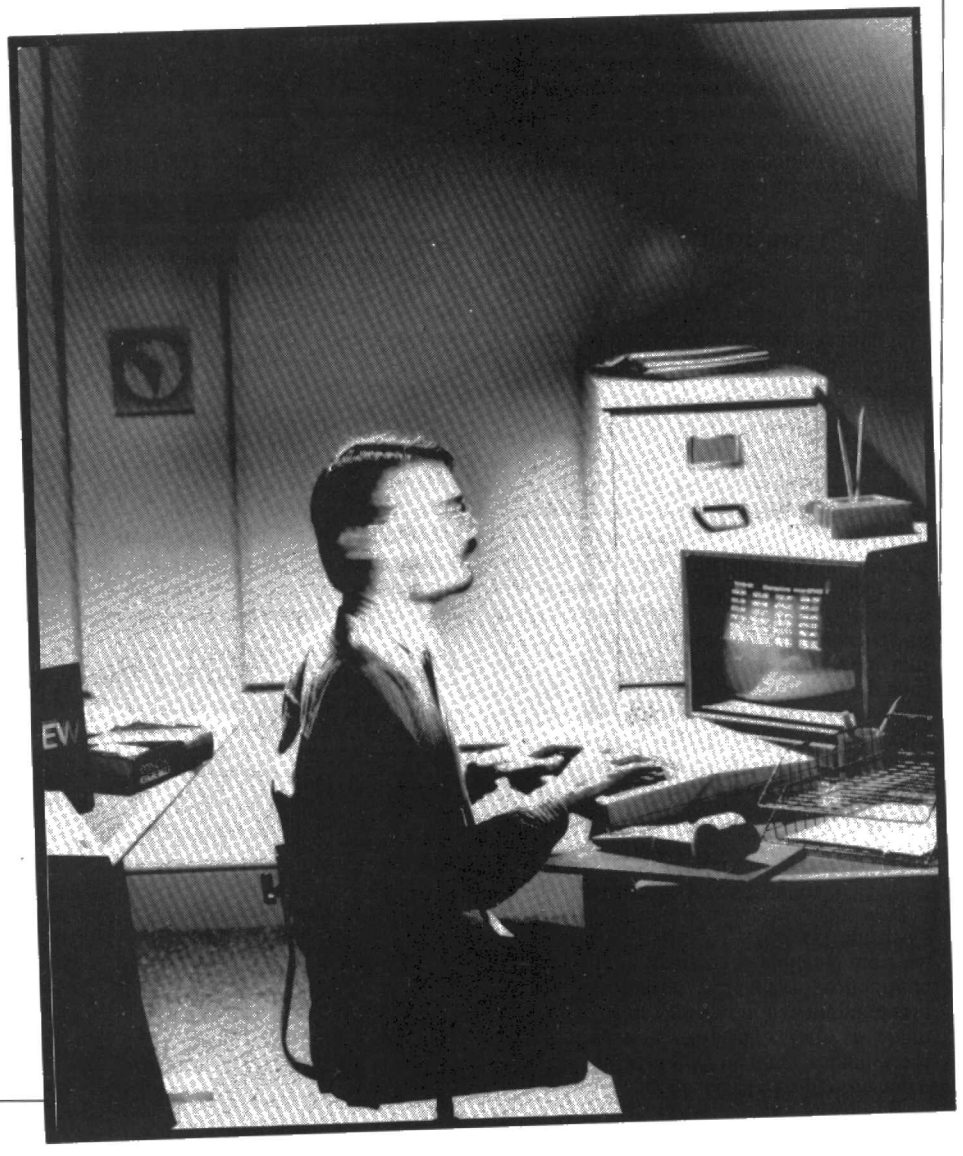
WHAT IS A BUFFER?

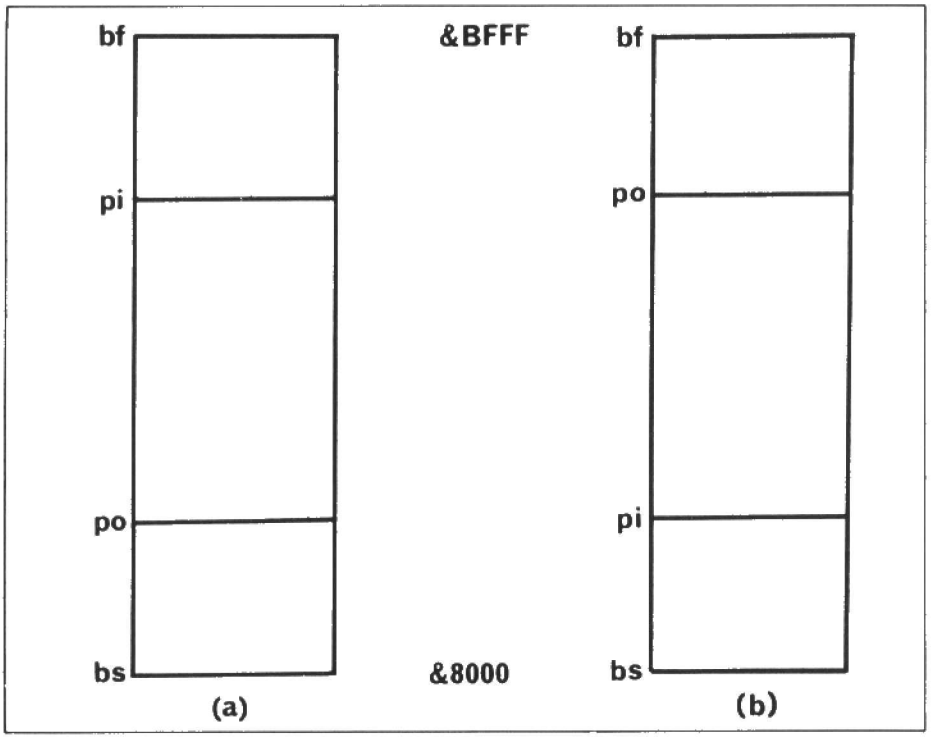
A buffer is in essence a mechanism to interface two independently operating units in a computer system. The two units may both be within one device as is the case with the microprocessor and the sound chip in the BBC micro or they may be in separate devices such as the microcomputer and a printer.

A common feature of both the examples is that the receiving device, the sound chip or printer, is much slower in operation than the sending device which is the microprocessor in both cases. The function of the buffer is to take information quickly from the fast sender so that it may continue on to the next task and then to pass the information to the slow receiver at whatever rate it requires. The effect is that the fast sender does not have to wait for the slow receiver to process the information before continuing.

Clearly, the size of the buffer plays an important part in determining the performance of the system. Imagine that the buffer was just one byte long and that the slow receiving device processes data one byte at a time. The first byte which goes into the buffer will be sent immediately to the slow receiver for processing and the buffer will be empty again. If the next byte comes into the buffer before the receiver has processed the first byte, the buffer will become full and the next byte from the sender will not be put into the buffer, so that the sender will have to wait until the slow receiver has processed its first byte and then taken the next byte from the buffer thereby emptying it, so that the sender may insert its next value and continue. Thus the sender is ultimately forced to go at the slow receivers speed when the buffer becomes full.

Naturally the bigger the buffer the longer it will take to fill and the greater is the likelihood that the sender will come to a part of its program where data is not being sent to the slow receiver, allowing this device to empty the buffer ready for the next onslaught of data from the fast sender. Choice of buffer size is usually a compromise between the desire to use big buffers to speed system operation and the shortage of available RAM memory.





Figures 1 (a & b). Input and output pointers move in RAM as indicated.

receive a character. These are called <po> for pointer output and <pi> for pointer input respectively. The two flags are the buffer empty flag <bef> and the buffer full flag <bff> and are set to a value of one if the appropriate condition is true, otherwise they are set to zero.

When the buffer is first activated and after it is purged the buffer empty flag is set, the buffer full flag is clear and <pi> and <po> are equal to <bs>. When a character is inserted into the buffer it is placed at the address contained in the input pointer <pi> which is then incremented by one. If the input pointer now equals the buffer finish address <bf> then it is made equal to the buffer start address <bs> thus providing address wrap around for the buffer. The input pointer is now compared with the output pointer and if they are equal the buffer full flag is set. Finally the buffer empty flag is cleared if it was set.

Character removal is similar in that the character at the address contained in the output pointer <po> is removed and the pointer is incremented by one. Again, if it equals the buffer finish address <bf> it is made equal to the buffer start address <bs>. The output pointer is now compared with the input pointer and if they are equal the buffer empty flag is set. Finally the buffer full flag is cleared if it was set.

The input and output pointers thus chase each other round and round the available RAM as shown in **Figures 1a** and **1b.** When used initially the pointers will be as shown in **Figure 1a** with <pi> bigger than <po> and the characters to be removed lying between <po> and <pi>. However, after wrap around, the pointers will be as shown in **Figure 1b** with <po> bigger than <pi> and the characters to be removed will lie from <po> to <bf> and from <bs> to <pi>. So long as <pi> and <po> are not equal there must be some characters left in the buffer waiting to be removed. The two pointers may become equal in either an insertion or removal operation. If equality occurs in an insertion operation then the input pointer has raced away from the output pointer, folded round the top of the RAM and approached the output buffer from behind. The buffer is thus full. On the other hand if equality occurs during a removal operation, the output pointer must have caught up the input pointer from behind and the buffer will thus be empty.

Buffers in the BBC Micro

The BBC micro has a multitude of input and output buffers to aid system operation and the printer buffer has an identification number of three. These buffers are not controlled individually but through three operating system routines. The first is for inserting a character into a buffer, the second is for removing a character from a buffer and the third is for counting the number of characters or spaces in a buffer

"Software follows normal sideways ROM protocol".

and for purging it. To make life easy Acorn have routed each routine through a vector. These are at &22A, &22C and &22E respectively. Thus to insert a character into a buffer, a jump should be made to the code whose address is contained at &22A and &22B with the character to be inserted in the accumulator and the buffer number in the X register. To remove a character from a buffer, a jump should be made to the code whose address is contained at &22C and &22D with the buffer number in the X register. On exit the next character to be removed from the buffer will be in the X register. To determine the number of characters in a buffer the carry and overflow flags should be clear and the X register should contain the buffer number before jumping to the code whose address is contained at &22E and &22F. On exit the X and Y registers will contain the result. To determine the number of spaces in a buffer the above process should be repeated with the carry flag set and to purge a buffer the overflow flag should be set. Anyone who seeks more complete details of these processes should refer to 'The Advanced User Guide For The BBC Micro' by Bray Dickens and Holmes.

Implementation

The software follows the normal sideways ROM protocol for a service ROM having one entry point at hexadecimal address &8003. It responds to three service calls. These are the '*HELP' service, the 'unknown star command' service and the 'claim private workspace' service.

When a '*HELP' service is received with no parameter then the ROM name is printed together with a list of the *HELP parameters to which it will respond. These correspond to the star commands which the ROM will recognise. If a parameter is sent with the star command eg '*HELP BUFFER' then the parameter is checked against the list to which the ROM will respond and if a match is found the corresponding message will be printed. The code to perform these tasks starts at <help> at line 7990 in the listing.

The unknown star command service is generated when the operating system fails to recognise a star command and the sideways ROMs are then given the chance to see whether they recognise it. This code is similar to the *HELP code except that instead of printing a message when a match is found the processor will jump to an action address containing code to perform the particular star command. The service code is located at <comm> at line 7280 in the listing and the table of commands and action addresses is <ctable> at line 7790. The action addresses of the commands bear the same names as the commands so that the code for *BUFFON is located at <buffon> etc.

The most complicated code is for *BUFFON. First, a check is made to ensure that the sideways RAM buffer is not already active. If it is then an error message is printed. Next, a check is made to determine whether any sideways RAM is present. This is effected by copying the code at <codeloc> down to the bottom of the stack area at &100 and then running it. This code selects each sideways ROM socket in turn starting at number 15 and determines whether RAM is present. When RAM is found, the routine returns with the socket number in the X register. If no RAM is found, a value of &FF is returned which causes an error message to be printed.

Next a check is made through the count/purge vector at &22E, to ensure that the original BBC printer buffer is empty, if not, an error message is given and the command is abandoned. If the buffer is empty then the code at <substart>, the intermediate code, is copied into the BBC buffer RAM at &880.

This code is of fundamental importance to the operation of the buffer since it enables the buffer control software in EPROM to read and write to the sideways RAM with the minimum of effort. The code has two entry points, one at &880 for read sideways RAM and the other at &883 for write sideways RAM. The code is copied into the BBC buffer for two reasons. The first is that it is able to access the sideways RAM which the controlling software cannot do directly. This is because it is a sideways ROM and is de-selected whenever the sideways RAM is activated to allow an access. Thus it is impossible to have the sideways RAM and the controlling EPROM activated at the same time, hence the use of intermediary code which will disable the controlling EPROM, access the sideways RAM and then reselect the controlling EPROM and pass on the requested information. The second reason for storing the intermediary code in RAM is that the controlling software can 'customise' it to suit the environment in which it is running. This customisation is confined to two aspects of the code operation. The first is that the slot number of the sideways RAM may vary from machine to machine and must be determined at the time when the controlling software is run. The sideways RAM slot number is inserted into the intermediary code at <ss3 + 1>, line 6830, so that the value which will be loaded into the Y register will not be &FF, which is an arbitary 'filler' value, but will be the actual slot number of the sideways RAM. The second customisation concerns the addresses in the LDA and STA instructions at <ss4> and <ss5>. These two byte addresses are again fillers because the actual addresses will not be known until the control software is run. In fact, the pointers <pi> and <po> are kept in these locations so that when the intermediate code is entered at &880 (read) the byte at <po> will be returned whilst when the code is entered at &883 (write) the byte in the accumulator will be stored at <pi>. It is the responsibility of the sideways ROM controlling software to increment <pi> and <po> and to assign them initial values.

The read and write routines in the intermediate code are used by the *BUFFON command to determine the amount of RAM present and to set up the values of <bs> and <bf>. Finally, if all subsequent checks are in order, the extra code for operating the buffer is connected to the operating system at the three buffer vectors described earlier. The connection is effected by the code <convecs> at line 2680. Operation of the vectoring system which Acorn have implemented on the BBC micro deserves an article in its own right but, briefly, each two byte vector between the addresses &200 and &235 may be directed at any memory location in any paged ROM even if it is not the currently active ROM. To activate this advanced vectoring mechanism the two byte vector at address &200 + 2*i should be changed to point at memory location &FF00 + 3*i and then the address in the paged ROM and its ROM slot number should be stored at &D9F (O.S. 1.2 only) + 3*i. Thus the contents of the insert vector at &22A should be saved in a convenient place and then the vector should be changed to &FF3F. Next the new buffer code address should be inserted at &DDE and &DDF with the ROM slot number at &DE0. The reason for this complicated vectoring system is to ensure that sideways ROM software will not de-activate itself when it calls any system function and to allow large amounts of additional code to be added to the operating system without taking up any additional space in the active memory space of the micro.

After the additional buffer handling code has been connected a flag (the top bit of the private workspace pointer) is set to show that the extended buffer is in operation and this ends the *BUFFON command.

The *BUFFON command simply reverses the vector connection procedure and restores the original values to the three vectors. *PURGEON and *PURGEOFF set or clear a flag which is examined whenever a printer buffer purge command is received. *PURGE resets <pi> and <po> to <bs>, sets the buffer empty flag and clears the buffer full flag.

"The performance of the BBC micro is greatly enhanced by the provision of a printer buffer facility".

After the vectors have been changed, buffer operations for all buffers will come to the controlling software in sideways ROM at locations <inscode>, <remcode> and <cntcode>. These examine the buffer number in the X register and if it is not 3 (indicating a printer buffer operation) then the original operating system code is run using the addresses which were saved during the vector connection procedure. If it is a printer buffer operation then the new code is run and the operating system code is not activated.

Software

The software is provided in the form of a BASIC 2 listing. You can tell which type of BASIC you have by pressing the BREAK key and typing REPORT <RETURN>. If the message reads (C)1981 Acorn you have BASIC 1, if (C)1982 Acorn you have BASIC 2. Readers with BASIC 1 will not be able to run this program and should see the end of this section. If the program is entered exactly as printed then it will run at values of PAGE up to &1900. A trap has been inserted at line 110 to prevent the program crashing if there is not enough space.

When the program is run, a listing is produced of the machine code instructions and at the left hand side of each line will be the execution address of that instruction. After the listing, each group of 128 bytes will be added up and compared with a reference value which was determined from a working program. If an error is found then this will be reported together with the start and finish addresses of the block in which the error occurred. The program should then be re-run, examining the addresses at the left hand side of the screen, to determine which instructions are included in the incorrect block. The lines in the BASIC program containing these instructions should then be carefully checked. A disadvantage of these crude checksums is that an error in one block may cause all subsequent blocks to appear faulty when in fact they are correct. The best procedure is thus to correct one error at a time and to rerun the program after each error is found. It is best to achieve a fully working error free program before attempting to customise it in any way. The ESCAPE key default state can be changed so that it will purge the printer buffer by changing <poff2> to <pon2> in line 2660.

The authors are prepared to supply copies of the program on tape, 40 or 80 track disc for £8 including p&p. Please state which you require. Readers with BASIC 1 will be provided with a machine code file ready for use in sideways RAM or for programming into an EPROM if this is requested when placing their order. The authors will also program EPROMs which are sent to them with the machine code output from the listing for a similar amount. Please send orders to: *Mr. B. V. Alderwick, 40 The Chase, Cashe's Green, STROUD, Gloucester GL5 4SB.*

Conclusion

This program provides an inexpensive way of making use of sideways RAM as a large printer buffer. The performance of the BBC micro is greatly improved by this facility which allows large amounts of text to be printed without the microcomputer having to wait for the printer. Happy programming!

Please Note

Unfortunately, lack of space has prevented us from including the program referred to. This will be featured next month but, for readers anxious to get started, please send us an sae together with a 50p PO to cover costs. Copies may also be obtained from the authors as shown above.

SINCLAIR'S QL

The machine behind the image

We asked Mike James to take a long, hard look at the QL and to assess the machine in the cold light of day some nine months after the hype that surrounded its launch.

The QL's biggest handicap is having been produced by Sinclair Research! There was a period not so long ago when any new product of Sir Clive's company was duly hailed as a technical, marketing or design achievement. Now the tide has turned and you can hear the knives being drawn even before the product is available for review.

Undeniably some of this change round is due to the misleading delivery dates promised for a number of recent products. The first was the infamous missing Spectrum Microdrive and this set the stage for the even more infamous missing QL . . . Events such as these and the general attitude towards Sinclair that has been building up over the past months makes it very difficult to come to a clear opinion on the QL.

Certainly it cannot live up to its pre-launch publicity and many early reviews of the machine were solely concerned with this failure. Later reviews seem to have become locked into the notion that "the QL is an incomplete bug ridden machine" and quote crazy obscure bugs in order to prove their point or resort to criticising the keyboard. (These reviews seemed so set on discussing the keyboard that they should be forever known as 'rubber keyboard reviews'!)

". . . not a super MPU".

Now that the QL is far enough away from its launch date to be available in a complete and final version it seems time that it was re-evaluated without reference to delivery delays, dongles or its early advertising literature. This is of course the intention of this article; to describe both the good and the bad points of the QL and most importantly to try and assess their effect on the usefulness of the machine. However, this article is not a complete review – there have been too many of those already – in that it assumes you know roughly what a QL looks like and few other salient details.

What's a 32 bit machine?

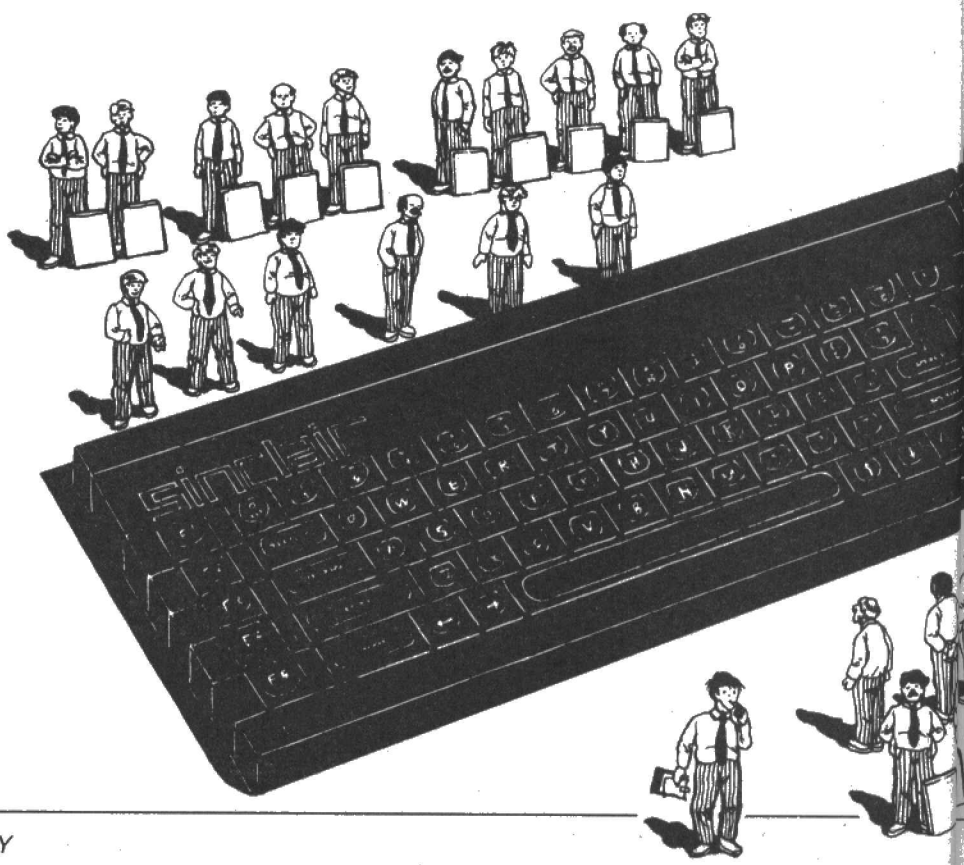
Perhaps the most important claim for the QL is that it is the first 16- or even 32-bit machine selling for hundreds rather than thousands of pounds. While any breakaway from the old fashioned Z80 or 6502 MPUs is to be encouraged, using the 68008 on its own doesn't produce a super computer.

The 68000 family of microprocessors is certainly a quantum leap in terms of its instruction set and architecture but it consists of a range of devices varying in processing power, from the 68008 up to the 68010 (or even the soon-to-be-produced 68020). While the 68000 or the 68010 deserve to be called 16- or 32-bit processors the 68008 used in the QL is rather a different case. Certainly it works internally with 16- and 32-bit quantities but it fetches and stores data in the familiar eight bit chunks. This slows processing down considerably and although the 68008 can be programmed as if it was a fully fledged 68000 it is closer to the 6809 in performance. (Some assembly language programmers even claim that the 6809 is faster and uses less memory than the 68008 in most applications!)

Therefore, although the 68008 is a member of the 68000 family, it is not a super MPU. It doesn't, for example, have enough power to make it worth trying to share it between a number of users or even between very many tasks. What it does offer is a very good instruction set and an ability to address a large amount of memory and these are not insignificant advantages.

Massive memory

Although the 68008 can address up to one Megabyte of memory the QL has been designed to run 128K of RAM as standard with the option of adding extra to bring the total up to 640K. The most interesting thing about the QL's memory capacity is the way that the standard 128K looks set to free users from any worries about running out

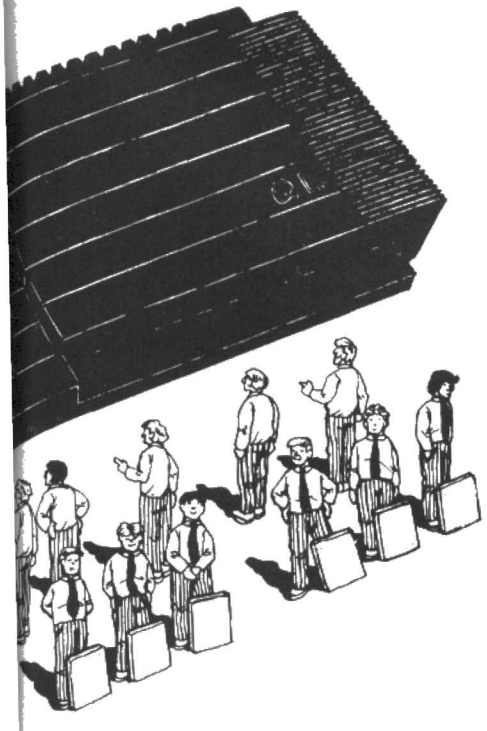


of storage space for programs etc.

The real situation is rather different! Having 128K of RAM to play with has obviously gone to the heads of the Sinclair hardware people. Instead of the economical use of memory found in the Spectrum's colour graphics, the QL uses 32K of memory for a full high resolution bit-mapped display. There is no dobut that the quality of display produced is very good – either 256 x 256 in eight colours or 512 x 256 in four colours with no restrictions on colour placement, but the price is also very high. After taking 32K out of the RAM the QL is left with only 96K which is beginning to look a lot closer to the 64K that most other computers start off with! A rough and ready calculation and a few simple tests suggests that the amount of memory available to Super BASIC programs and data is indeed around 64K. This is very good but it is only half of the advertised RAM capacity and is closer to the amount of RAM that other machines offer BASIC than you might expect!

Other odd features of the QL's memory map are the large areas committed to other than RAM expansion. Using 48K of resident ROM and allowing 16K for ROM cartridges is reasonable enough but the QL has around 320K of address space allocated to I/O devices! I can only suppose that this waste of address space is because Sinclair is planning to use the cost cutting trick of not fully decoding all of the address lines to any given peripheral so creating multiple images of its control and data registers through the address space. In computing, no matter how much memory a machine has there is always an application or a program that needs more and the gaps in the QL's address space are bound to make someone angry sooner or later.

"... having 128K of RAM to play with has gone to Sinclair's head ... SuperBASIC must make do with only half the advertised RAM capacity ..."



Slow graphics

Having commented on the way that the 68008's address space has been used it is worth returning to the 32K of RAM allocated to the video display. High resolution video displays are very attractive and the QL has one of the best, but consider the problem of manipulating the screen to produce animation effects. The video RAM is organised so that two bytes hold the colour codes of four pixels in low resolution mode or eight pixels in high resolution mode. If you want to work with eight colour 8 x 8 sprites then storing the colour codes for a sprite takes 32 memory accesses. A similar figure results if you decide to work in the higher resolution four colour mode because of the need to use more pixels horizontally to maintain the overall squareness of the sprite.

These 32 memory accesses should be compared to the eight needed for the same task (in fewer colours) on the Spectrum Clearly, while it is possible to produce animation on the QL, the increased amount of memory that you have to move almost cancels out any speed advantage of the 68008. Without doing any detailed calculations I would put the overall graphics performance at about the same speed as the Spectrum but of course you do get an increase in both the resolution and the number of colours you can use.

Such a large amount of video memory also takes time to block move as can be seen when the QL scrolls or pans a screen. Indeed the fact that scrolls and pans take roughly .2 of a second to complete is a proof that the 68008 is good at rearranging 32K of RAM!

Peripheral opinions

When it comes to communicating with the outside world the QL's collection of peripherals are about as controversial as you can find. The notorious unresponsive keyboard has been mentioned in every review since the QL was first not quite produced. The shocking truth is that in practice the keyboard is quite reasonable! Certainly if you are used to a quality keyboard, and most reviewers are because of their need to use a word processor, then you will find the QL keyboard something of a disappointment – it doesn't feel quite right. However this feeling soon passes over and before you know where you are you can type very efficiently and confidently using it. The current attitude is simply a reflection of the fact that opinions about keyboards are very much a matter of what you are used to.

Similarly, reviews have criticised the Microdrives to the point of saying that they are unusable lumps of plastic either because they are slow or because they are unreliable or both. This is not, in my experience, at all true. The QL Microdrives are reasonably fast and, what is more important, their access times do not vary much with the file size or with the number of files on a cartridge. This is a subtle point but what it means is that if you try to load a very small test program then the Microdrive will whirr for a few seconds as it tries to find the program and then it will whirr for a second as it loads it. As a result of this the reviewer concludes that Microdrives are very slow and misses the point that even if the test program had been large the time to find it would have been the same and the time to load it wouldn't have added that much.

As regards reliability the Microdrives are not perfect – in the course of developing a large number of programs they did occasionally lose a file. This level of unreliability can be overcome by using the second Microdrive to take backup copies and is no worse than that encountered when using tape based systems.

As to Microdrive cartridges, these are very delicate but, as long as they are treated with care, they do seem to last.

Super Software?

The QL comes complete with rather a lot of software – SuperBASIC and QDOS in ROM and the four Psion programs – Quill, Abacus, Easel and Archive on Microdrive cartridges. Even though, in time, other software houses are bound to produce vast quantities of QL software it is likely that the programs given away with the machine are going to dominate. In other words people will buy the QL as a package of hardware and software rather than just purchasing it and then going out to choose the best text processor for it etc. This makes the quality of the free software all the more important for the success of the QL.

SuperBASIC is the star of the QL's software and it really is super! As a dialect of BASIC it is difficult to beat for its control structures, procedures and its ability to be extended without having to use machine code. This is not to say that SuperBASIC is perfect, there are several omissions and a number of awkward ways of doing things but on the whole these can be easily overcome. For example unlike most dialects SuperBASIC doesn't include the SGN function but this can be added with a few lines of SuperBASIC.

Perhaps the major problem area in SuperBASIC is its handling of graphics. The use of pixel, graphics and character co-ordinate systems is very confusing and the manual is distinctly unhelpful when it comes to clarifying the position. For example the SCALE command can be used to set the maximum y value in any window but the manual simply states that the maximum x co-ordinate will be set so that a circle still appears circular. This is very reasonable but it is quite difficult to work with a graphics system when you only know the limits on one of the axes! There are ways round this problem, for example when the vertical scale is 100 the horizontal scale is 148 and knowing this at least allows you to work out the horizontal scale for any vertical scale. On the plus side the availability of windows is a great advantage but it is not new – the BBC micro supports any number of independent text and graphics windows. On the whole Super BASIC's handling of graphics is not as slick as that offered by say BBC BASIC but it's reasonable enough when you finally manage to understand it.

From the programmer's point of view one of the real advantages of Super BASIC is the ease with which you can add to its built-in commands. All procedures are incorporated into the language as if they were new commands and user defined functions are indistinguishable from supplied functions. It is not until you make use of this facility that its potential is fully appreciated.

QDOS

The impact of QDOS on the average QL user is slight. It is a multi-tasking operating system, there's an example in the manual to prove the point, but none of the QL's application software takes advantage of this. To be more precise there is no way of running more than one Super BASIC program at a time (unless someone is being very secretive about how to do it). The part of QDOS that the user does see is mainly concerned with I/O, and on this count QDOS is fine as it adopts a Spectrum-like approach to devices and channels. For example QDOS is responsible for all text output to any screen windows that have been defined.

One of the biggest problems awaiting any games designer planning to use the QL is its lack of any sensible way of defining graphics characters. It is true that you can change any part of the character set but each character is only 5 x 9 pixels and has a fixed border of paper pixels. This makes it very difficult to put a number of characters together to form a solid shape. However, there is nothing stopping you from adding a standard user defined graphics facility based on an 8 x 8 pixel character to SuperBASIC. (If you'd like to know how then consult "The QL Gamesmaster" by Kay Ewbank, Mike James and S M Gee, published by Granada where this approach is adopted and explained.)

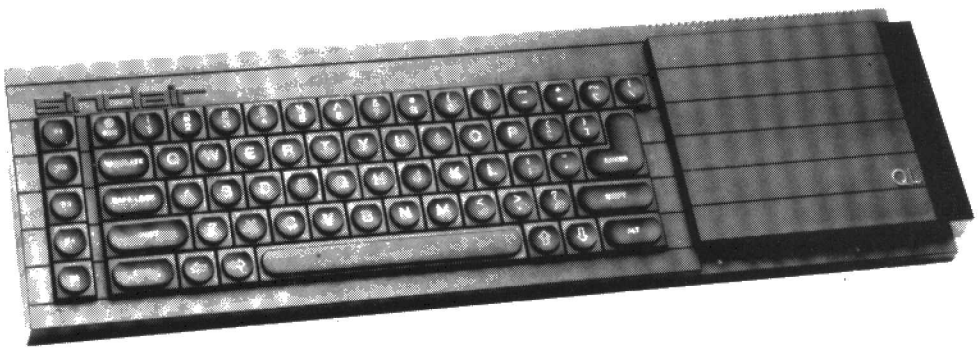
QDOS is also responsible for the keyboard input and in this case its multi-tasking nature can cause something of a problem. It appears that keyboard input and screen output are treated as two separate tasks under QDOS and this means that while QDOS will read keypresses even while the QL is apparently busy doing something else, the exact time that the corresponding character appears on the screen is up to QDOS. This lack of direct association between pressing a key and the letter appearing on the screen is slightly unnerving and it is probably another reason for the mistaken criticism of the QL keyboard. Apart from this the facilities that QDOS provides for handling data from many different devices in a uniform fashion is a welcome change from the usual confusion.

"... in many ways the QL is a capable and versatile machine still in search of its software ..."

The four Psion packages have attracted almost as much attention as the QL itself. Usually the comment "nice software shame about the machine" has been in the back of most reviewers minds but in fact the software is at best only average. Of the four Archive and Easel are possibly the best in terms of innovation. The former is a powerful, flexible and easy to use database program that ranks with the best eg dBase 2. Unfortunately it is let down by the speed of the Microdrives but nevertheless it is usable for real applications. Easel is a business graphics program that allows for programmers to construct display graphs suitable for presentation at sales drives, conferences, etc; it is easy to use and very versatile but it is not a statistics package so don't expect it to do more than it was intended to – graphs and charts are its limit.

"More like a quantum hop than leap".

As for the others, Abacus is a very standard spreadsheet program with some good points although it takes up so much memory that there is only 15K of work space left. Quill is a very acceptable word processor, it is not as sophisticated as WordStar and its imitators but it is remarkably easy to learn how to use, which should be considered a major advantage. However it is very slow. If you can type even a little you will not find it difficult to get a line or two ahead of the screen display, it's not that what you type is lost it just takes time to appear. The same thing applies to the operation of commands, even simple ones such as cursor movement. This behaviour is a consequence of the independence of screen output and keyboard input mentioned earlier. It is to be hoped that Psion will look into this sluggish response and do something about it.



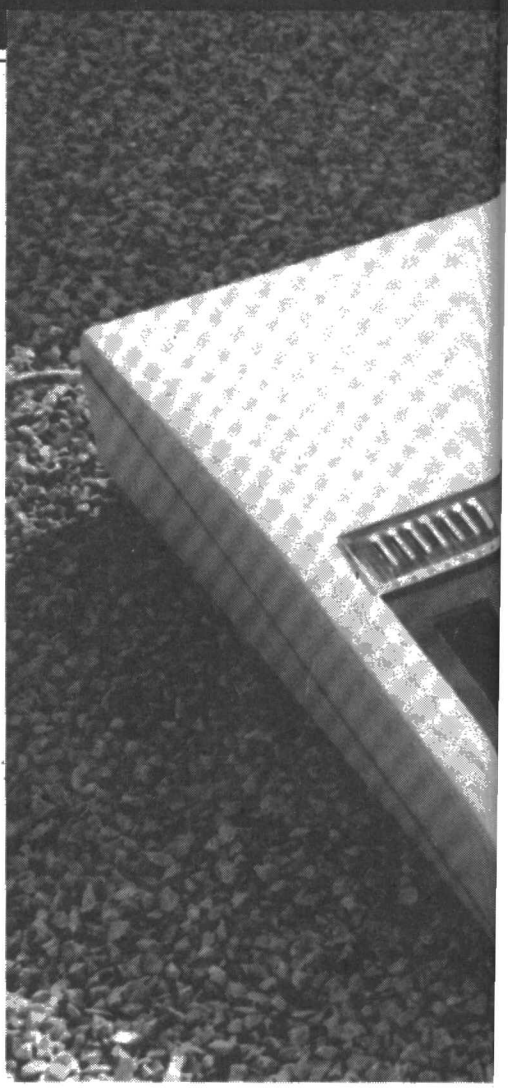
A Quantum Leap?

As you can deduce from the above, in my opinion the QL is most definitely not a Quantum leap – at the most it is a small Quantum hop. It is different; it's not based on a traditional microprocessor but as a member of a family with a future, it's broken the 64K memory barrier and it has low cost built-in storage devices that are not tape based. Put this hardware specification together with the software and what you get is not a super computer but something that before the QL you couldn't buy for even twice the money.

If you are interested in programming in BASIC then SuperBASIC is excellent and the QL highly recommended. If you are interested in pure applications based on the four packages then things are not so clear cut – it depends which of them you would have to use. In many ways the QL is a capable and versatile machine still in search of its software.

BBC RANDOM ACCESS

Adam Denning returns to the techniques of random access filing on the BBC micro with a look at some of the finer points involved.



Now that we've discovered how to use random access files from a record structured point of view, we need to consider some of the finer points. Suppose we open a file with ten records or arbitrary length and then take away a few of the records. Suddenly we have a file that is bigger than it needs to be and there doesn't seem to be any way of altering this. We can't assign to EXT# as it gives us a syntax error and we can't position the pointer at what we consider to be the end of the file and then close it, as the system just ignores our efforts. We have to come to a compromise between minimum file size and disc storage efficiency.

There are three main ways of approaching this problem. A value which all null records will take may be decided upon so that, on reading the record back, we find that it holds invalid data and should thus be ignored. Unfortunately, this uses a lot of disc space as each null record occupies as much space as a valid record. Another way would be to hold the number of each record in the first few bytes of the file, so that if the first 100 bytes were defined as holding the numbers of the records in the file we could store up to 100 records but they wouldn't have to be sequential. The problem now is that, once we've decided upon how many records our file is going to hold, it cannot accommodate any more until our programs are changed and the file is altered drastically.

The best manner of approach is to maintain a rudimentary linked list. Here the first byte (or two bytes, if we intend having more than 255 records) of each record would hold the number of that record. A null entry would mean that the space occupied by this record is, in fact, empty and should be used to hold the next written record. Thus it is unlikely that we will ever have more than one blank record in our file at any one time, although if we deleted a number of records and only wrote a few back, we would temporarily have a large inefficient space. This linked list method would inevitably mean that reads and writes could not automatically jump to the required part of the file, as each record would have to be examined to find the right one. This means a large number of disc accesses and a deterioration in performance.

There is a way round this too, of course. If we keep a 'map' of the file in memory which is updated after every read or write, we can find our record and go to it straight away. After all it's a lot faster searching through a few bytes in memory than it is to look through an equal number of records on disc. If we were really efficient we could write this record map to disc as a separate file each time we use our program, and then each subsequent use would only need to read the record map from disc in one go rather than constructing it before the program runs. This does, of course, require setting up first.

We would have an algorithm something like this:

record_file = file containing the physical records

record_map = file containing the last record map

The record map consists of:

2 bytes number of records in file
then 2 bytes per record, containing record number if valid or &FFF if not.

So,

Initialisation (the first run)

Create null length file **record_file**

Create 2 byte long file record_map containing two zeros.

Subsequent runs

Read **record_map** into memory
Open **record_file** for random access

Read: find record number in record map (memory copy)
set record_file pointer to record number * record length
read relevant record

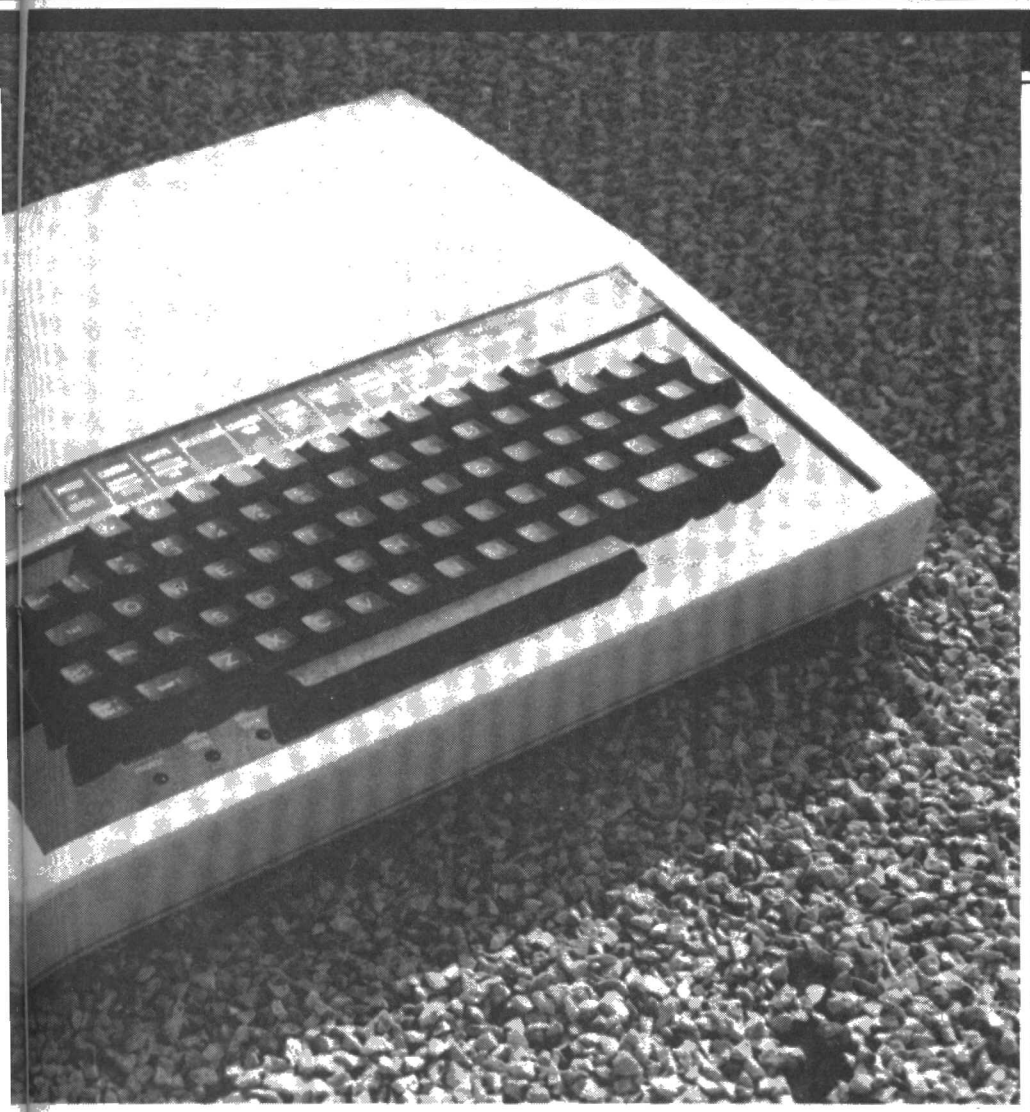
write: find record number in map or first invalid record number (&FFFF) – if record non existant AND no free records, then record number = number of records in file; increment number of records in file
set record_file pointer to record number * record length
write record
If area was previously unused then poke record number into relevant slot in record map in memory

repeat until end of program

Write back record_map to disc
Close record_file.

The initialisation part is easy enough:

This is very simple; it uses OSFIND to open and close the requisite files and



OSBPUT to send two bytes of zero to the record map file. It must be entered with **wherever%** holding the requisite location for the code, **file$** holding the name of the record file and **map$** holding the name of the record map file. Obviously if the whole program were written in machine code these variables would not need to exist.

The main part of the code is going to be written as four procedures – **PROCstart** to open record_file and to read record_map into RAM, **PROCread_record** to read a given record, **PROCwrite_record** to write a given record and **PROCfinish** to close record_file and save record_map back to disc. The first and the last are the easiest of the four, so we'll write these two first.

The first procedure needs to be called with **wherever%** holding the start location for the code, **file$** holding the name of the record file, **map%** holding the name of the map file and **here%** holding the address to which the record map is to be loaded. The second needs to be entered with the file handle returned by OSFIND for **record_file** in Y. It also needs to know the address from which record_map is to be saved, held in **here%.**

Both procedures use standard operating system calls described in the User Guide, but notice that the information given about OSFIND is slightly wrong. The file handle is returned in A, *not* in Y! The next installment will describe PROCread_record and PROCwrite_record, and then we'll turn the whole thing into pure machine code. Finally, we'll think of an application for it all!

PROC Init

```
1000DEFPROCinit
1010FOR C%=0TO2STEP2
1020P%=wherever%
1030[OPTC%
1040.inithere
1050LDA #&80
1060LDX #record_file MOD 256
1070LDY #record_file DIV 256
1080JSR &FFCE \OSFIND
1090TAY
1100BEQ error
1110LDA #0
1120JSR &FFCE \OSFIND
1130LDA #&80
1140LDX #record_map MOD 256
1150LDY #record_map DIV 256
1160JSR &FFCE \OSFIND
1170TAY
1180BEQ error
1190LDA #0
1200JSR &FFD4 \OSBPUT
1210JSR &FFD4 \OSBPUT
1220JMP &FFCE \OSFIND
1230.error
1240BRK:BRK
1250EQUS"File error!"
1260BRK
1270.record_file
1280]
1290$record_file=file$
1300P%=P%+1+LENfile$:record_map=P%
1310$record_map=map$:P%=P%+1+LENmap$
1320NEXTC%
1330CALLinithere
1340ENDPROC
```

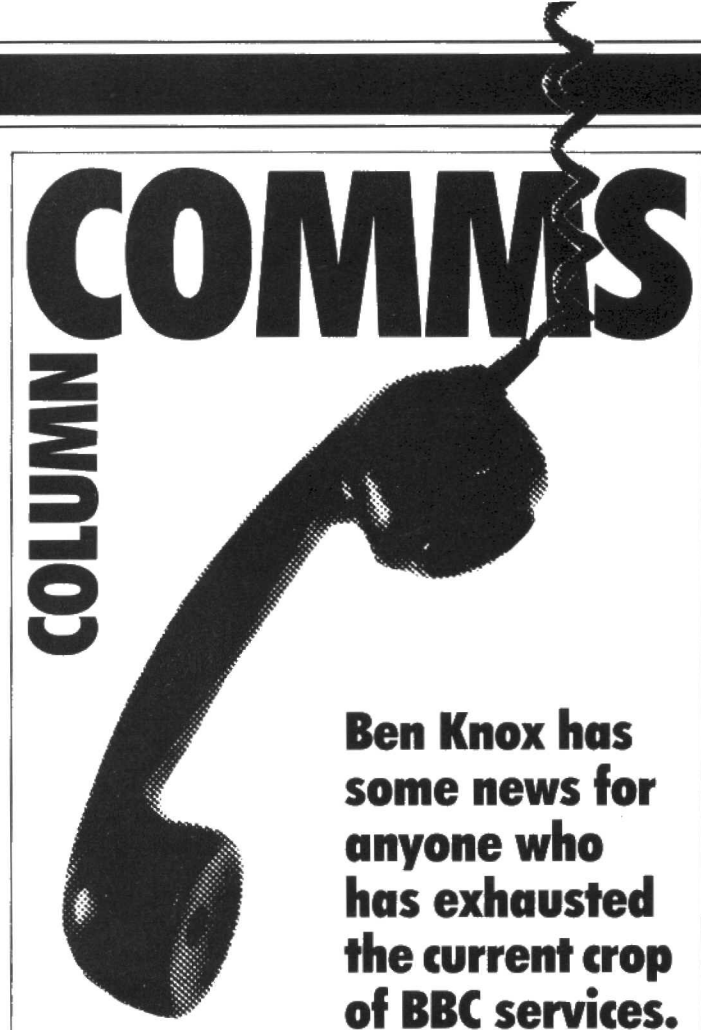
PROC Start

```
2000DEFPROCstart
2010FORC%=0TO2STEP2
2020P%=wherever%
2030[OPTC%
2040.start
2050LDA #&C0
2060LDX #record_file MOD 256
2070LDY #record_file DIV 256
2080JSR &FFCE \OSFIND
2090TAY
2100BEQ error
2110PHA
2120LDX #record_map MOD 256
2130STX osblock
2140LDX #record_map DIV 256
2150STX osblock+1
2160LDX #osblock MOD 256
2170LDY #osblock DIV 256
2180LDA #&FF
2190JSR &FFDD \OSFILE
2200PLA
2210TAY
2220RTS
2230.error
2240BRK:BRK
2250EQUS"File error!"
2260BRK
2270.record_file
2280]
2290$record_file=file$:P%=P%+1+LENfile$:record
    _map=P%
2300$record_map=map$:P%=P%+1+LENmap$:osblock=P%
2310osblock!2=here%:osblock!6=0
2320NEXTC%
2330CALLstart
2340ENDPROC
```

PROC Finish

```
3000DEFPROCfinish
3010FORC%=0TO2STEP2
3020P%=wherever%
3030[OPTC%
3040.finish
3050LDA #0
3060JSR &FFCE \OSFIND
3070LDX #record_map MOD 256
3080STX osblock
3090LDX #record_map DIV 256
3100STX osblock+1
3110LDX #osblock MOD 256
3120LDY #osblock DIV 256
3130LDA #0
3140JMP &FFDD \OSFILE
3150.record_map
3160]
3170$record_map=map$:P%=P%+1+LENmap$
3180osblock=P%:osblock!2=0:osblock!6=0
3190osblock!10=here%
3200length%=(?here%+256*here%?1)*2+2+here%
3210osblock!14=length%
3220NEXTC%
3230CALLfinish
3240ENDPROC
```

COMMS COLUMN



Ben Knox has some news for anyone who has exhausted the current crop of BBC services.

Once you have exhausted either the list of Bulletin Board Systems (BBS) given on the next page, or worn your finger to the bone from repeatedly dialling a seemingly endlessly engaged BBS, the more adventurous might be interested in investigating some commercial systems.

One of the problems about using such systems is that often long distance calls are required when you want to access them. Anyone who has received a telephone bill since starting in the computer communications game, will know the shock finding that the amount is two or three times more than the usual figure – and that is just from making calls to local Bulletin Boards.

Perhaps the new 'Midnight-line' service from British Telecom could keep the bills down. For a quarterly payment of about one hundred pounds all local and long-distance calls which are made between midnight and 6.00 am are free. However this service obviously won't help much for international calls to, say, America

PSS

One solution to this is the 'Packet Switch Stream' (PSS). PSS is a data network, similar to a normal telephone service, except that it has been designed only to be used by computers or terminals.

By splitting data up into small quantities (packets) and intermingling it with data from other sources information can be transferred from one location to another both efficiently and cheaply. The costs are less than using the public telephone system because instead of one whole telephone line being used for one information transaction, a PSS line can be used for many transactions simultaneously.

There are PSS exchanges in all the major cities in Britain so, for most people, each one is only a local call away. Since its introduction in 1981, many companies, universities and other organisations have had their mainframe computers linked to PSS. PSS will therefore allow us to use many systems both inside and outside the country fairly cheaply.

To use PSS, you will need to get hold of a 'Network User ID' (NUI), a twelve character string by which the PSS system can recognise you as a 'bona-fide' user. These NUIs are available from the address given at the end of this article. Having obtained your NUI, you need only dial your PSS exchange to have access to almost any large computer system in the world, so long as you know its 'Network User Address' (NUA). This is like its telephone number, you must enter it when requested and you will be put through.

Some useful PSS NUAs

Essex University (for Multi-User Dungeon, a big adventure game):	A2206411411
CompuServe Information Service:	A93132
The Source Information Utility:	A93110301000240 0 or A93110301001590 0
Telecom Gold:	A219201004

Once you have been registered, you will be able to use PSS from the date given when you receive your NUI. For some reason people who have been recently registered on PSS haven't received any information on how to use it, so here is what you need to do to get started.

Making the right connection

Connect your computer to the telephone with your modem. Dial the number of the PSS exchange which you have been registered on. The call will be answered by a high pitched tone. Switch you modem online and put your communications software in Terminal mode. When the modem carriers have locked (the carrier light goes on) type: Carriage Return (<CR>) twice, followed by the letter D and the number 1.

You should now receive something like: LON/AO2-19201059. The first three letters are the exchange identifier, in this case London, the rest of the characters specify the port to which you have been assigned.

You must now enter your NUI, which is, say, 'ECMPSSOOOXXX' (this is not a real NUI!). To enter it, type N followed directly by the NUI, so in this case: NECMPSSOOOXXX. The last six characters will not be displayed on your screen as a security precaution. Now type <CR>.

The prompt 'ADD?' will be displayed. This is where you type in the NUA of the computer which you want to call. NUAs have two parts, the country identifier and the actual number of the computer. For example, the full number for the PSS online directory, called 'Hostess', is: 234 21920100515. To enter this, and address at the prompt, you have to type: A2192010051544, omitting the country code (234) and adding A. You will now be automatically connected to the system which you have called.

International calls always start: A9 . . .,. For example, a system called CompuServe is accessed by typing: A93132. CompuServe is the US equivalent of Prestel, only bigger (and, in some parts, better) and will be covered in a future article.

When you have finished using the system, you may be disconnected automatically. If this is the case you will see: CLR DTE(00):00:00:03:25: 376:28. This tells you how long you were connected (3 minutes and 25 seconds), how many data 'packets' have been received (376) and how many were sent by you (28).

If the system you have called does not use auto-log-off, you may terminate the connection by typing: <Cntrl> P (control and P keys pressed together) followed by CLR<cr>. You will get the same details about connect time and amount of data transferred as before except it will be preceeded by CLR CONF(00) this time.

It will cost you a one-off fee of £25 to be registered on each PSS exchange. Thereafter, there is a quarterly rental charge of £6.25. Inland charges are, at Standard rate: £1 per hour of connection time and 25p per 'kilo-segment' (about 64Kbytes of data). Cheap rate: 90p per hour and 15p per kilo-segment.

International calls are more expensive and depend upon which country you want to reach. As a guide, a call to the USA costs £6.00 per hour and £3.50 per kilo-segment. This compares well with direct dial telephone charges: one hour will cost £36 at peak rate.

Further Information about PSS may be obtained from: *PSS Sales, Customer Service Group, GO7 Lutyens House, 1-6 Finsbury Circus, LONDON EC2M 7LY. Tel: 01-920 0661.*

BULLETIN BOARD SERVICES

Name	Number	Notes (check times before calling)
BABBS	(0742) 667983	Times: 24hrs. British Apple System User Group
BBCBBS	(01) 579 2288	Times: 24hrs. BBC 'Micro Live'
CABB	(01) 631 3076	Times: 24hrs. Computer Answers magazine. 300 & 1200/75 baud.
CBBS London	(01) 399 2136	Times: Sun 1700-2200
CBBS Surrey	(04862) 25174	Times: 24hrs
CBBS SW	(0626) 890014	Times: 24hrs
City BBS	(01) 606 4194	Times: 24hrs 1200/75 Weds
Distel	(01) 679 1888	Times: 24hrs Commercial (free) BBS run by Display Electronics
Estelle	(0279) 443511	Times: Office hours only. STC customers BBS
Forum-80 Hull	(0482) 859169	Times: Mon-Fri 1700-2330; 1200-2330 weekends
Forum-80 London	(01) 902 2546	Times: 1900-2200 wkdays; 1200-2200 wkends. Ring and ask for Forum-80
Hamnet	(0482) 497150	Times: 1800-0800. Radio Ham BBS.
Liverpool Mailbox	(051) 4288924	Times: 24hrs.
Mailbox-80 W. Midlands	(0384) 635336	Times: 1800-0800. Ring back
Manchester BBS	(061) 4273711	Times: Sun-Thu 2230-0000; Fri 2330-0200; Sat 2230-0200
Maptel	(0702) 552941	Times: 24hrs Maplin BBS Tele-ordering & stock levels
Microweb	(061) 4564157	Times: 24hrs Micro User magazine
NBBBS	(0827) 288810	Times: 24hrs Ring back
Southern BBS	(0243) 511077	Times: 2000-0200 Ring back
Stoke ITec	(0782) 265078	Times: 24hrs Remote CP/M
TBBS Blandford	(0258) 54494	Times: 24hrs
TBBS London	(01) 348 9400	Times 24hrs
TBBS Nottingham	(0602) 289783	Times: 24hrs Multi speed: 300/300, 1200/75 baud
TBBS Southampton	(0703) 437200	Times: 24hrs

CBM 64 I/O PORT

Robert Penfold describes a versatile I/O port that also incorporates two 16-bit counter/timers.

For user add-ons the Commodore 64 is undoubtedly one of the best home computers currently available. It has a versatile user port that provides eight input/output lines plus three lines intended primarily for handshaking purposes. Two 16 bit counter/timers are also accessible via this port, which is similar to (but not the same as) the BBC model B machine's user port.

One of the most attractive features of the CBM 64 from the user add-on point of view is its cartridge port, an interface which has received scant attention. Although primarily intended to accept plug-in program cartridges, it also enables further input and output lines to be easily added to the machine. It is, in fact, comparable to the much better publicised 1MHz Bus of the BBC model B computer, and gives similar scope for expansion. The only drawback is that connections to it must be made via a 2 x 22 way 0.1 inch pitch male edge connector, which is fine for a cartridge port but is not the most convenient type of connector for a general purpose expansion port. However, where just a simple expansion port is required, it is simply a matter of building it on a double-sided PCB having a suitable built-in edge connector, so that the board plugs into the port in much the same way as a program cartridge.

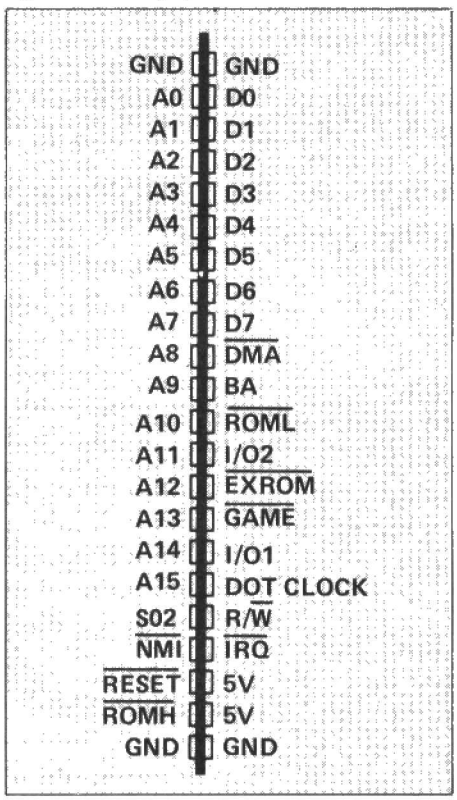
This is the system adopted with this expansion port, which uses a 6821 PIA (parallel interface adaptor) to provide 20 input/output lines. These consist of two 8 bit ports, both of which have each line individually programmable as an input or an output, plus two useful handshake lines per port.

Connection details of the CBM 64's cartridge port are provided in **Figure 1.** The full data, address, and control buses are available, together with a number of other lines which are mostly concerned with switching in plug-in ROMs and switching out the internal RAM in the address space occupied by these ROMs. However, there are two other lines which are analogous to the NPGFC and NPGFD lines of the BBC computer's 1MHz Bus. These are I/O 1 and I/O 2, which are decoded from the eight most significant lines of the address bus, and go low when any address from 56832 to 57087 and 57088 to 57343 (respectively) are accessed. In other words these are activated by accessing addresses in pages &DE and page &DF respectively, giving some 512 addresses for expansion purposes if these outputs are decoded together with the eight least significant address lines. Alternatively, if only one or two add-on circuits are required these outputs can be used without having to resort to any further address decoding whatever.

6821 PIA

Having ready-decoded outputs makes it very simple indeed to add a PIA to the port. In fact, as can be seen from the circuit diagram of this port, the only component needed is the PIA itself plus the input and output plugs. The data bus plus the R/W, RESET, and clock lines of the port couple straight through to the corresponding lines of the 6821 The CBM 64 has a 980kHZ clock frequency, and the standard 1MHz 6821 is perfectly adequate for IC1. The negative chip enable input of IC1 is fed from line I/O 2 of the cartridge port, while the RS0 and RS1 (register select) inputs are fed from address lines A0 and A1. This places the port at the four addresses from 57088 to 57091. In fact the registers appear at addresses throughout the 57088 to 57343 address range of I/O 2, and no other devices can be placed in this address range. However, the full 256 addresses of I/O 1 are left unused for possible further expansion.

Figure 1. Cartridge port connections.



The CBM 64 does, of course, have all its 64K of address space occupied by RAM. This does not lead to a conflict between this port and the RAM though, since the RAM is automatically disabled when either I/O 1 or I/O 2 is active.

The 6821 requires a single 5 volt power supply, and a suitable power output is available at the cartridge port. The total maximum allowable current drain from the cartridge and user ports is 450 milliamps, which leaves about 400 milliamps spare for other add-on circuits. The 6821 has two positive chip enable inputs (pins 22 and 24) that are not needed in this case, and they are tied to the positive supply rail.

The two ports are designated "A" and "B" by the chip manufacturer. PA0 to PA7 are the eight data lines of port A, while CA1 and CA2 are the handshake lines. Similarly, PB0 to PB1 are the data lines of port B and CB1 plus CB2 are the handshake lines.

The 6831 has two IRQ interrupt outputs, but it is highly unlikely that these would be needed in this case, and they have not been implemented here.

Construction

Figure 3 provides details of the double sided PCB. Start by fitting Veropins or through-pins at the fifteen points where through-board connections are required. If you use Veropins these should be trimmed almost flush with the board prior to soldering them in place. IC1 is a MOS device and a (40 pin DIL) IC socket should therefore be used for this component. Do not plug it in place until the rest of the board has been completed. The two ports are ten to 20 way IDC plugs. The "right angled" type are used on the prototype, but the "straight" type will also fit onto the board and can be used if preferred.

The completed board simply plugs into the cartridge socket, with the side of the board containing IC1, PL2, and PL3 uppermost. **Figure 5** shows the connections to PL2 and PL3.

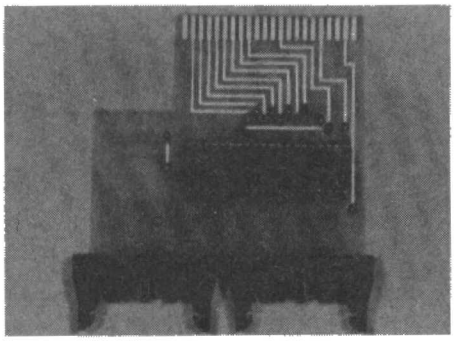
Programming

In order to use the port in even a simple application it is essential to have a basic

understanding of the way in which the registers of the 6821 are used. There are six registers at the four addresses occupied by the device, and the table given below shows how this system operates.

Address	Control Register Bit 2	Register Selected
57088	1	Port A
57088	0	Data Direction Register A
57089	Not Relevant	Control Register A
57090	1	Port B
57090	0	Data Direction Register B
57091	Not Relevant	Control Register B

The data direction register is used to set each line of the relevant port as either an input or an output. Writing 1 to a bit sets the corresponding in/out line as an output; writing a 0 to a bit sets the corresponding in/out line as an input. For instance, writing 15 (00001111 in binary) to data direction register A sets PA0 to PA3 as outputs and PA4 to PA7 as inputs. However, before data can be written to a data direction register bit 2 of the relevant control register



must be set low. This bit must then be set high again to enable data to be written to or read from the port. All the registers are set at zero by a reset pulse at switch-on, but in practice it is advisable not to assume that any registers are already at zero just in case a glitch occured at switch-on.

As a simple example of setting up a port, to set PA0 to PA7 as outputs the following commands would be used:

POKE 57089,0 (gives access to data direction register A)
POKE 57088,355 (sets all port A lines as outputs)
POKE 57089,4 (gives access to port B)
POKE 57088,x (writes value x to port A)

When used as inputs the handshake lines are edge sensitive. In other words they are activated by either a high to low or a low to high transition, depending on the mode that is programmed. An active transition sets a bit of the appropriate control register high (bit 7 for CA1/CB1, but 6 for CA2/CB2). CA1 and CB1 are controlled by bits 0 and 1 of the relevant control register, but as we are not using the interrupt capability of the 6821 in this case it is only bit 1 that is of interest. It sets the operating mode in the following manner:

Bit 1 at 0 High to low transition sets CRA/B to bit 7 high
Bit 0 at 0 Low to high transition sets CRA/B bit 7 high

The CA1 flag is reset by reading port A or writing to it, and the CB1 flag is reset by reading from or writing to port B.

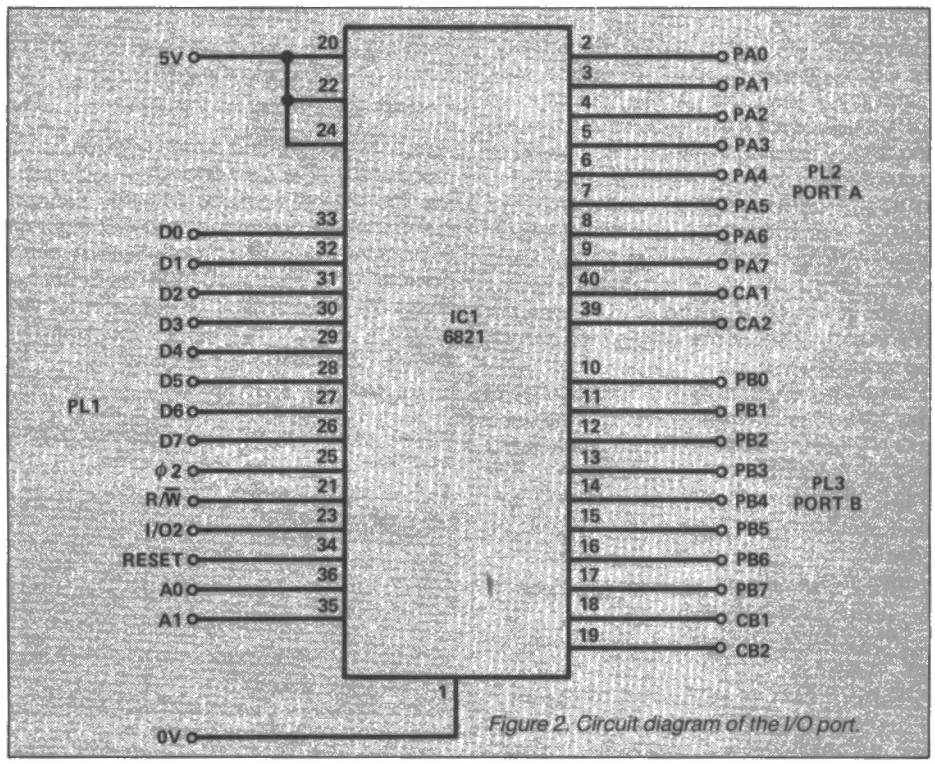


Figure 2. Circuit diagram of the I/O port.

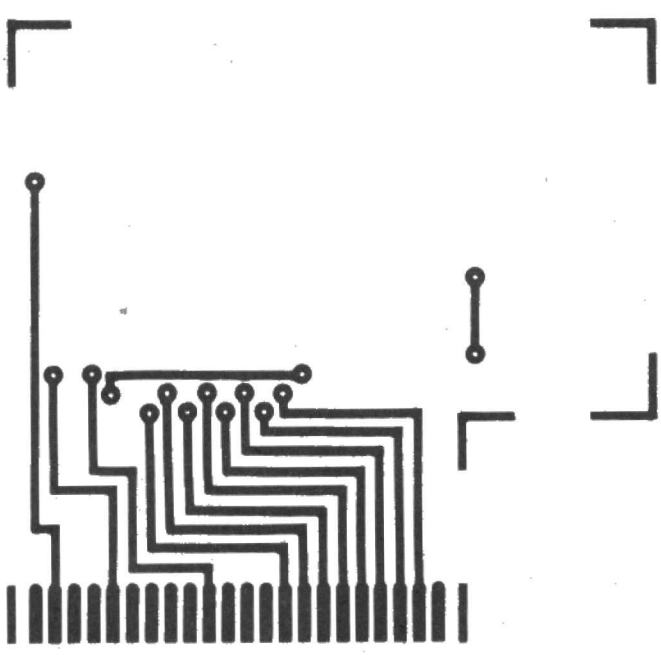


Figure 3a (left). Topside foil pattern.

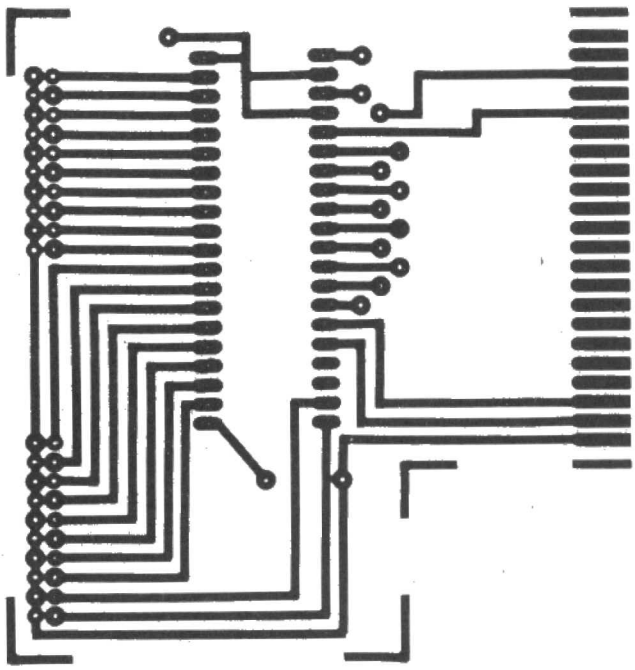


Figure 3b (right). Lower foil pattern.

The CA2 and CB2 handshake lines are a little more complex in that they can operate as inputs or outputs. They are controlled by bits 3 to 5 of the appropriate control register, and the table shown below gives details of the bit states required for the two input and four output modes that are applicable in this case.

Bit 5	Bit 4	Bit 3	Decimal number	MODE
0	0	0	0	High to low input
0	1	0	16	Low to high input
1	0	0	32	Handshake output
1	0	1	40	Negative pulse output
1	1	0	48	Low output
1	1	1	56	High output

The input modes are the same as for CA1 and CA2, and the flags are reset in the same way. In the handshake output mode CA2 is set low by an active transition of CA1 and is set high again by reading port A. CB2 operates in much the same way, but it is set low by an active transition on CB1, and it is reset to the high state by writing to (not reading from) port B. The pulse output mode gives a short negative pulse of about 1us in duration on CA2 following a read operation at port A, or on CB2 following a write operation to port B. This mode is useful for automatically providing a strobe pulse. In the other two output modes the line is simply set continuously high or low, as required.

As a simple example of setting up one of the ports, assume that port B is to be used as an 8 bit output that must provide a negative strobe output pulse and a high to low input handshake input. The following setting up procedure for the 6821 would be used.

POKE 57091,0 (gives access to data direction register B)
POKE 57090,255 (sets all port B lines as outputs)
POKE 57091,44 (gives access to port B by. setting bit 2 high, sets bit 1 low to give high to low input on CB1, and sets bits 3 and 5 high to give pulsed output mode on CB2).

Data would then be POKEd to address 57090, while the CB1 input flag would be used as bit 7 of the control register at address 57091. WAIT is a useful but often overlooked instruction in the CBM 64's BASIC, and it is ideal for monitoring a flag. All that this instruction does is to wait (looping the program indefinitely) until a certain bit or one of several specified bits of the given address goes high. The number following the address indicates which bit or bits are to be monitored. In this case the program could be made to wait for the CB1 flag to go high using the instruction:

WAIT 57091,128

What this command actually does is to monitor the specified address, logic ANDing the returned values with the second number in the command, and looping until a non zero result is obtained. A third number (the inversion) can be used to invert the specified bit or bits, so that the program loops until a certain bit or one of several bits goes low. For instance:

WAIT 57088,128,128

would loop until bit 7 at address 57088 went low.

PARTS LIST

IC1	6821
PL1	Part of PCB
PL2,3	20 way IDC plugs (2 off)

Printed circuit board; Veropins or through pins; Solder.

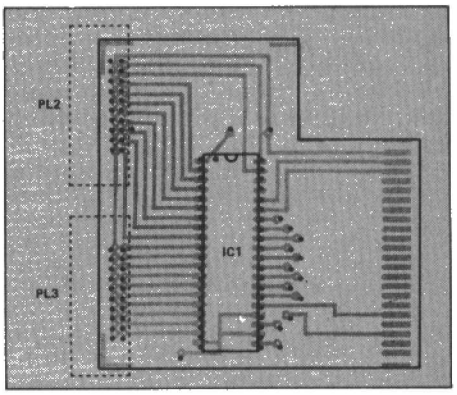


Figure 4. The I/O ports component overlay.

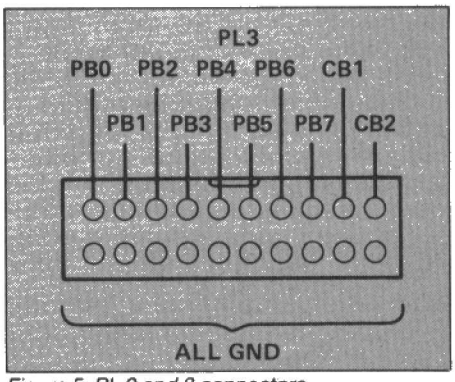
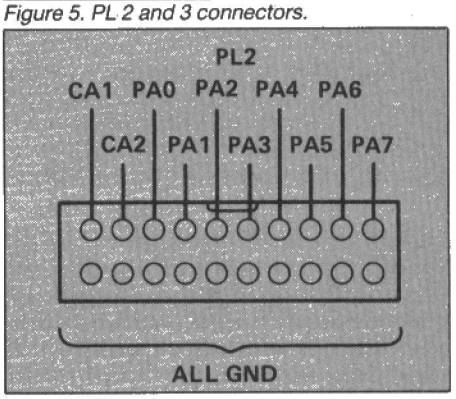
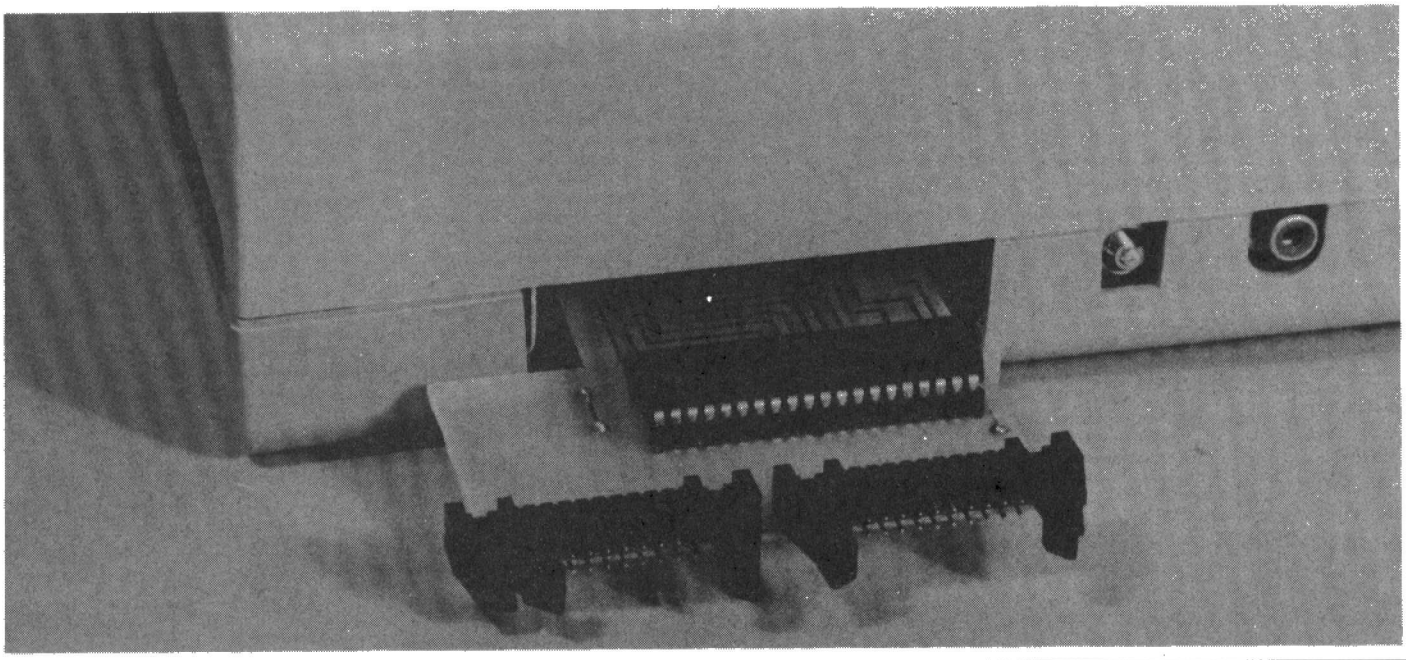


Figure 5. PL 2 and 3 connectors.



The completed project is plugged into the CBM 64's cartridge socket with the side of the board containing IC1, PL2 and PL3 uppermost.



HIGH SPEED BEEB INTERFACE

Paul Beverley continues his series on BBC micro interfacing with a discussion of the software techniques necessary to make the most of the speeded up 1MHz bus.

Last month I explained how to speed up the 1MHz bus and the User and Printer Ports to 2MHz. This month the task is to illustrate the software techniques needed to make best use of these extra high speed interfaces, by looking at the task of generating waveforms using an external digital to analogue converter. Also included in this article are the modifications necessary to make the frequency meter program from the September 1984 issue of *E&CM* work with the higher speed interface.

Waveform generation

To illustrate some of the programming techniques of high speed interfacing we will look at ways of getting the computer to produce waveforms. The simplest to produce are square waves: assuming you want TTL voltages (0 – 5 volts) no external hardware is necessary, just a connection to the User Port. For more complex waveforms we will need to have some form of digital to analogue converter attached to the computer, such as the 'DAC-Pack' add-on to DCP Microdevelopments' "Interbeeb" interface which connects to the 1MHz bus.

Square waves

Let us look first at square waves: simply making one of the port lines go up and down as fast as possible. The fastest you can do this in Basic is less than 1 kHz and so we will move straight on to machine code. A simple machine code loop, such as that shown in **Listing 1,** on a normal 1MHz VIA will produce a square wave on the PB0 line at 100 kHz (10 microseconds period). Using a 2MHz VIA will improve this to 111 kHz (9 microseconds).

The problem with this technique is that the microcomputer is tied up to generating the square wave and cannot do anything else at the same time. It is possible however to get the VIA itself to generate a square wave independently of the processor which means that normal processing can carry on as the square wave is being produced. This is done by putting Timer 1 into the free-running mode as shown by the program in **Listing 2.**

The number N% is entered into Timer 1 as two bytes, and the timing, as was explained in the July issue of *E&CM,* is such that the VIA produces a square wave with a period of 2*(N%+2) cycles. The value of N% can be zero, thus the smallest period is 4 cycles, and therefore the ordinary 6522 with its 1MHz clock can produce square waves up to a maximum of 250 kHz (4 microseconds) while the 6522A at 2MHz can give us a square wave at 500 kHz.

LISTING 1

```
 10 DIM code 20
 20 ?&FE62 = 1 : REM Set PB0 as output
 30 P% = code
 40 [ SEI
 50 .loop
 60 INX
 70 STX &FE60
 80 JMP loop
 90 ]
100 CALL code
```

LISTING 2

```
 10 PB  = &FE60
 20 ACR = &FE6B
 30 T1L = &FE64
 40 T1H = &FE65
 50 ?ACR= 192
 60 REPEAT
 70   INPUT N%
 80   ?T1L = N% MOD 256
 90   ?T1H = N% DIV 256
100   UNTIL 0
110 END
```

More complex waveforms

If we want to produce more complex waveforms then we need to use an external DAC. I shall assume that you are using the Interbeeb DAC Pack and use the addressing appropriate to that, but if you have a DAC on the User port or Printer port or at a different address on the 1MHz bus then all you need to do is to change the address accordingly.

The voltage output of the DAC is proportional to the single byte number stored in it. Thus to produce a periodic waveform you need a program which loops round, going through a particular set of data, repeatedly putting the different bytes of data out to the DAC in sequence.

Probably the simplest way of doing this is to divide the waveform up into 256 steps and put the 256 data values in one page of memory (whose address is data%) and then use:

```
SEI
.loop
LDA data%,X
STA dac%
INX
JMP loop
```

This picks up one particular value from the data table and outputs it to the DAC, increments the X register to point to the next piece of data, and then jumps back ready to pick up that bit of data. The SEI instruction at the beginning of the program sets the interrupt flag in order to stop interrupts occurring as these would cause the waveform to be unstable.

If we look at the timing of this program, we find that it takes 7 microseconds (of 6.5 microseconds on the 2MHz 6522A) to go once round the loop and output one single step. This gives a resultant wave of period 256 x 7 = 1792 microseconds which is a frequency of 558Hz (601Hz on 6522A). This program forms a closed machine code loop so that the only way out is to press the break key, but you could program the break key with *KEY10 OLD:M to get the program back again. We could avoid this problem by putting an extra statement into the loop which checked the state of one of the I/O lines to decide whether to continue in the loop, but since we are concentrating on getting as much speed as possible, we will ignore this idea.

When using this method it is important to be sure that the data table is in a single page of memory and does not cross a page boundary ie you can use &2000 to &20FF, but if you use &2080 to &217F the timing goes astray, because when LDA &2080,X crosses a page boundary the instruction takes one cycle longer. This means that one half of the wave will occupy a slightly longer time than the other.

In order to increase the speed of the pro-

LISTING 3. Function generator program.

```
 10 REM Function Generator Program
 20 REM Working Version - 9.10.84
 30 REM (C) Norwich Computer Services
 40 MODE6
 50 PROCinitialise
 60 PROCassemble
 70 REPEAT
 80   CLS
 90   REPEAT
100     INPUT"Steps per cycle",S%
110     IF S%>X% CLS:PRINT"Maximum ";X%
120     UNTIL S%<=X%
130   H%=S%/2
140   REPEAT
150     PROCmenu
160     INPUT "Waveform Number ",W%
170     IF W%=6 INPUT'"Pulse width as a % of the
        period",E%
180     A%=Q%-5*(S%-1)
190     M%=A%+1
200     FOR N%=S%-1 TO 1 STEP-1
210       IF W%=1 ?M%=FNsine
220       IF W%=2 ?M%=FNsquare
230       IF W%=3 ?M%=FNtri
240       IF W%=4 ?M%=FNsaw
250       IF W%=5 ?M%=FNfunny1
260       IF W%=6 ?M%=FNpulse
270       IF W%=7 ?M%=FNfullwave
280       IF W%=8 ?M%=FNhalfwave
290       IF W%=9  W%=0:end%=TRUE
300       IF W%=10 W%=0
310       IF W%>10 W%=-1
320       IF W%<1  N%=0
330       IF N%MOD10=1 PRINT N%-1:VDU11,13
340       M%=M%+5
350       NEXT
360     IF W%>0 CALL G%
370     UNTIL W%=0
380   UNTIL end%
390 END
400
410 DEF PROCinitialise
420 VDU19;4;0;
430 dac%=&FCC3
440 end%=FALSE
450 X%=2000
460 ENDPROC
470
480 DEF PROCassemble
490 PRINT "Assembling"
500 DIM P% (X%*5+40)
510 FOR N%=1TOX%
520   [OPT 0
530   LDA #0
540   STA dac%
550   ]
560   NEXT
570
580 [OPT 0
590 .Q%
600 LDA &FCC1
610 BNE P%+5
620 JMP (&404)
630 CLI
640 RTS
650
660 .G%
670 SEI
680 JMP (&404)
690 ]
700 ENDPROC
710
720 DEFFNsine
730 =128+127*COS(2*PI*N%/(S%-1))
740
750 DEFFNsquare
760 IF N%<H% V%=0 ELSE V%=255
770 =V%
780
790 DEFFNtri
800 IFN%<H% V%=N%*255/H%+3 ELSE V%=255-((N%-H%+2)*255/(H%+2))
810 IF V%>255 V%=255
820 =V%
830
840 DEFFNsaw
850 =N%*255/S%
860
870 DEFFNfunny1
880 IF N%<255 =N%
890 IF N%<600 =128+127*COS(PI*(N%-256)/68)
900 IF N%<1000 =((N%MOD100)>50)*-100
910 IF N%<1400 =((N%MOD100)>50)*-155+100
920 =RND
930
940 DEFFNpulse
950 IF N%<E%*S%/100 =255 ELSE =0
960
970 DEFFNfullwave
980 =ABS(255*COS(PI*N%/(S%-1)))
990
1000 DEFFNhalfwave
1010 V%=255*SIN(2*PI*N%/(S%-1))
1020 IF V%<0 V%=0
1030 =V%
1040
1050 DEFPROCmenu
1060 VDU12
1070 PRINT" Waveform synthesiser"''
1080 PRINT" 1. Sine"
1090 PRINT" 2. Square"
1100 PRINT" 3. Triangular"
1110 PRINT" 4. Sawtooth"
1120 PRINT" 5. Funny1 (Steps > 1400)"
1130 PRINT" 6. Pulse"
1140 PRINT" 7. Full wave rectified sine"
1150 PRINT" 8. Halfwave rectified sine"''
1160 PRINT" 9. End the program"'
1170 PRINT"10. Change number of steps per cycle"''
1180 ENDPROC
```

gram we might think of creating a loop that consisted of 256 repetitions of the indexed load, the store and the INX, and then jump back to the beginning of that loop. This can be done by enclosing the assembly of these three instructions in a FOR/NEXT loop.

However if we have got an individual instruction for each load, why not use an absolute load from the data table as follows:

```
FOR N%=0 TO 255
[OPT 2
LDA data% + N%
STA dac%
]
NEXT N%
```

Each time round the loop, data% + N% calculates the absolute address of the next byte of data to be used.

This then leads to the next idea which is to put the data within the loop itself – and this is the fastest method of all. That is, by using the immediate mode of addressing for the load, since this only takes two cycles instead of the four taken by an absolute load. The program then consists of a number, indeed it can be ANY number, of repetitions of:

```
LDA #data_item
STA dac%
```

Using this method, each individual step in the waveform takes 6 cycles, which gives a time of only 3 microseconds on the 6522A (4 microseconds on the 1MHz 6522). This means for example that a 256 step waveform would have a frequency of 1.3 kHz. Another advantage is that the number of steps is entirely under the control of the programmer. For a higher frequency, you simply use fewer steps, although this of course coarsens the waveform. If you want a smoother waveform, you use more steps, but this then reduces the frequency.

Once the program has been assembled, the data within the program can be changed by poking new values into the memory locations which follow the LDA # instructions. A program, using this method, which can be used to generate all sorts of different waveforms of any length, is given in **Listing 3.** You should be able to use this program to produce whatever waveshape you like by writing the appropriate function.

One point to note about the program is that, in order to avoid having to reassemble the program every time the user decides to change the number of steps, the jump back to the beginning of the loop is indirect ie the integer variable A% (located at address &404) is calculated at line 180 to point to the position in memory for the start address which will give the requisite number of steps before the jump back is encountered. Also, the program is entered at address G%, where it meets an SEI instruction before jumping into the loop with the same JMP (&404) instruction which is used at the end of the actual loop.

In order to get out of the loop without pressing the break key, the program checks &FCC1 each time it is about to jump back. This is the address of the digital inputs on the Interbeeb unit. Thus if contact is made between the "COM" output

and any of the digital inputs, the DAC output stops and the program displays the menu again. An alternative to this is to use:

```
530  LDX #0
540  STX dac%
```

and add:

```
675  LDA #16
```

and change:

```
600  BIT &FE40
610  BEQ P%+5
```

What this does is to check fire button 1 on the joystick input. LDA #32 would allow you to use the other fire button instead.

Finally, the number X% at line 450 sets the maximum number of steps possible per loop. This is controlled because for each step you need 5 bytes of machine code program. As it stands, there is enough memory available for just over 3000 steps (roughly 15k bytes of machine code program!), but if you wanted more, you could go into mode 7 and change line 450 as appropriate, but the larger the value of X% you use, the longer it takes to assemble the machine code at the very beginning of the program. If you choose too large a value for X% you will simply get a DIM space error at line 500.

Improved frequency meter

Improving the speed of the VIA to 2MHz makes possible an increased range on the frequency meter program given in the September issue. It can now range automatically from 0.1 Hz to about 960 kHz. One or two modifications to the program were necessary to account for the change in VIA clock rate, but also in the course of testing it out with the higher speed clock, one or two improvements were made to the 'PROCvIf' section. The original program line numbering has been maintained, and **Listing 4** indicates which lines that have been modified or added. The comments that were included in the original program have been omitted to save space, thus if you wish to understand the operation of the program you will need to refer to the September issue.

The section around lines 420 – 480 has been modified not only to account for the difference in VIA clock speed, but also to improve the performance at low frequency. It will now work down to less than 0.1 Hz. The only problem is that below 0.1 Hz the display goes to exponential form and the exponent is not displayed since the number string is truncated. If you really want to work below 0.1 Hz (=10 seconds per cycle = 10 ztreH?) then you will need to modify the display routine with:

```
625  IF frequency<0.1 value=
     frequency*1000
635  IF frequency<0.1 PRINT " mHz" :
     ENDPROC
```

according to the way you want it to be displayed.

LISTING 4. Frequency meter program from the September issue, with improvements and modifications for the 2MHz 6522A VIA.

```
1   REM Frequency Meter Program
2   REM 2 MHz Working Version – 9.10.84
3   REM (C) Norwich Computer Services

423 REPEAT
426   OK=1
430   IF NOT base% = base% – RND(8000)
433     R% = A%*20000 MOD base%
436     IF R%<T% OR R%>(base%–T%) OK=0
440     IF base%<T%+1 OK=0: base%=RND AND &FFFF
445   UNTIL OK
480 ti%=A% * 20000 DIV base% * base%
730 targetcount% = &1E000
750 T% = 15000
770 k = 2*K%*K%*fclock
```

MORE QL MULTITASKING

Adam Denning continues his series on the QL by taking a closer look at some routines for controlling jobs from SuperBASIC.

After last month's foray into QL multi-tasking, we mentioned that it was about time we gave ourselves some degree of control over jobs from SuperBASIC. The few routines here give us just that and only occupy 600 bytes of code. There are seven routines in all, allowing us to see which jobs are currently held in the machine and to do all sorts of things to them such as suspending, releasing and killing them. Other routines allow us to create and activate jobs and alter a job's priority. In all the routines requiring the ID of a specific job we have gone for the job number and tag convention rather than the entire 32 bit job ID, as this ID is only easy to remember in hex but us humans tend to be more familiar with decimal. These are the routines:

JOBS#channel

This procedure lists out all the jobs in the machine to the specified channel, in the format:

Job Tag Owner Priority Name

The job and tag are the individual components of the job ID, the owner is the job number of the job which created this job, the priority is a number between 0 and 127 and the name is the string starting at byte 8 of a job in standard QDOS format. If the job is currently suspended then the priority is preceded by an 'S'. To keep this code as compact as possible it takes advantage of various screen driver routines to move the cursor. Consequently the individual fields are not separated if the specified channel is not open a screen or console device, as these routines have no effect on non-screen channels.

info$ = CJOB$ (codelength,datalength [,startaddress])

This function attempts to create a job with the specified parameters. The third parameter would not normally be used unless a job is ROM resident, but can be passed to QDOS to tell it where the job lies in memory. If this parameter is not present then QDOS allocates its own area for the job in available memory. The string returned is always a 16 character hexadecimal string containing the tag of the

LISTING 1.

```
LOC       OBJECT      STMT  SOURCE STATEMENT

                        1 * Extensions to SuperBasic - procedures and functions
                        2 * Job control procedures
                        3 * By Adam Denning Copyright (C) 1984 Adam Denning
                        4
    =0001               5 MT_CJOB   EQU     1
    =0002               6 MT_JINF   EQU     2
    =0004               7 MT_FRJOB  EQU     4
    =000B               8 MT_SUSJB  EQU     8
    =0009               9 MT_RELJB  EQU     9
    =000A              10 MT_ACTIV  EQU     $A
    =000B              11 MT_PRIOR  EQU     $B
                       12
    =0005              13 ID_SBYTE  EQU     5
    =0011              14 SD_TAB    EQU     $11
                       15
    =00CE              16 UT_MINT   EQU     $CE
    =00D0              17 UT_MTEXT  EQU     $D0
    =0110              18 BP_INIT   EQU     $110
    =0112              19 CA_GTINT  EQU     $112
    =011B              20 CA_GTLIN  EQU     $118
    =011A              21 BV_CHRIX  EQU     $11A
                       22
    =FFF1              23 ERR_BP    EQU     -15
    =0001              24 RET_STR   EQU     1
    =0028              25 CH_LENCH  EQU     $28
                       26
    =0030              27 BV_CHBAS  EQU     $30
    =0058              28 BV_RIP    EQU     $58
                       29
0000' 43FA 0008        30 SET_UP    LEA.L   PROC_DEF,A1
0004' 3478 0110        31           MOVE.W  BP_INIT,A2
0008' 4ED2             32           JMP     (A2)
                       33
000A' 0006             34 PROC_DEF  DC.W    6
                       35
000C' 00CC             36           DC.W    JOBS_PROC-*
000E' 04               37           DC.B    4
000F' 4A4F 4253 00     38           DC.B    'JOBS',0
                       39
0014' 009C             40           DC.W    AJOB_PROC-*
0016' 04               41           DC.B    4
0017' 414A 4F42 00     42           DC.B    'AJOB',0
                       43
001C' 002E             44           DC.W    KJOB_PROC-*
001E' 04               45           DC.B    4
001F' 4B4A 4F42 00     46           DC.B    'KJOB',0
                       47
0024' 005E             48           DC.W    PJOB_PROC-*
0026' 04               49           DC.B    4
0027' 504A 4F42 00     50           DC.B    'PJOB',0
                       51
002C' 004E             52           DC.W    SJOB_PROC-*
002E' 04               53           DC.B    4
002F' 534A 4F42 00     54           DC.B    'SJOB',0
                       55
0034' 001E             56           DC.W    RJOB_PROC-*
0036' 04               57           DC.B    4
0037' 524A 4F42 00     58           DC.B    'RJOB',0
                       59
003C' 0000             60           DC.W    0
003E' 0001             61           DC.W    1
                       62
0040' 018B             63           DC.W    FN_CJOB-*
0042' 05               64           DC.B    5
0043' 434A 4F42 24     65           DC.B    'CJOB$'
                       66
0048' 0000             67           DC.W    0
                       68
                       69
004A' 610E             70 KJOB_PROC BSR.S   JOB2_COMM
004C' 7004             71           MOVEQ   #MT_FRJOB,D0
004E' 4E41             72           TRAP    #1
0050' 4E75             73           RTS
                       74
0052' 6106             75 RJOB_PROC BSR.S   JOB2_COMM
0054' 7009             76           MOVEQ   #MT_RELJB,D0
0056' 4E41             77           TRAP    #1
0058' 4E75             78           RTS
                       79
005A' 3478 0112        80 JOB2_COMM MOVE.W  CA_GTINT,A2
005E' 4E92             81           JSR     (A2)
0060' 6614             82           BNE.S   EXIT_JOB2
0062' 70F1             83           MOVEQ   #ERR_BP,D0
0064' 0C43 0002        84           CMPI.W  #2,D3
```

newly created job in the first four characters, the job number in the next four, and the job's start address in the last eight. Numerous functions exist to aid in the conversion of this string into decimal numbers as appropriate. The job will be created with the SuperBASIC interpreter (job 0, tag 0) as its owner.

AJOB job,tag, priority,timeout

This procedure activates a job with the specified initial priority. If the timeout is 0 then the command has the same effect as the last stage of EXEC, where the job specified is activated and then control is returned to the SuperBASIC interpreter. If the timeout is –1 the effect is similar to EXEC_W – the job is activated and control does not return to the SuperBasic interpreter until the specified job has terminated (by killing itself or by being killed). As activation only needs to be done once during a job's life, don't use this procedure on jobs which have already been activated (even if they have a priority of zero) – the effects can be rather interesting! Only use it after a job has been created with CJOB$.

KJOB job,tag

This procedure uses MT_FRJOB to kill a job and all its subsidiaries. It is very final and is a very good way of clearing out any jobs from memory which somehow failed to kill themselves.

PJOB job,tag, priority

This procedure is used to alter the specified job's priority to any value from 0 to 127.

SJOB job,tag,timeout

This suspends the specified job for the specified number of 20ms intervals between 0 and 32767. A timeout value of –1 suspends the job forever, or at least until it is subsequently deliberately released.

RJOB job,tag

This procedure releases a suspended job.

The code

The code for these routines is shown in **Listing 1.** Initially the file was assembled using a comprehensive header file similar to the one shown last month (mdv1_header_asm), but as this file is so large and takes up so much magazine space, the necessary equates have been included in the file itself. The code was assembled using the Metacomco assembler, but will of course assemble on most other assemblers except that not all of them allow labels to be unique in less than the first eight characters, which means that the labels EXIT_JOB2 and EXIT_JOB3 would need to be changed to EXITJOB2 and EXITJOB3.

As usual, we have the small set up code and procedure definition table, which together allow QDOS to include our routines in the SuperBASIC name list. Each procedure or function then starts, and is entered from SuperBASIC at the specified label. The routines can be split into various groups – those with two parameters, those with three parameters, and then the odd ones out. KJOB and RJOB fall into the first category, so they both call a subroutine called JOB2_COMM. This collects the arguments as integers using CA_GTINT and checks that they are valid. If they are then the two parameters (which are the job number and the tag) are combined to form the real job ID in register D1. This subroutine then ends and the individual routines load up D0 with the requisite QDOS manager trap keys, execute a TRAP #1 instruction, and return to SuperBASIC. This means that any errors (Invalid job, Not complete, or whatever) are returned straight to the SuperBASIC interpreter.

The routines requiring three parameters are SJOB and PJOB, and these both call a common subroutine called JOB3_COMM. This uses CA_GTINT to collect all three arguments and check them, and then if everything is okay the combined job ID is put into D1 and the third parameter is put in D3. On return to their respective routines, SJOB has its registers set up correctly so it just loads MT_SUSJB into D0, does a TRAP #1 and returns to Basic, while PJOB needs the third parameter, currently in D3,

LISTING 1 – Continued

```
0068' 660C          85            BNE.S   EXIT_JOB2
006A' 3236 9802     86            MOVE.W  2(A6,A1.L),D1
006E' 4841          87            SWAP    D1
0070' 3236 9800     88            MOVE.W  0(A6,A1.L),D1
0074' 4E75          89            RTS
0076' 221F          90 EXIT_JOB2  MOVE.L  (A7)+,D1
0078' 4E75          91            RTS
                    92
007A' 6110          93 SJOB_PROC  BSR.S   JOB3_COMM
007C' 7008          94            MOVEQ   #MT_SUSJB,D0
007E' 4E41          95            TRAP    #1
0080' 4E75          96            RTS
                    97
0082' 6108          98 PJOB_PROC  BSR.S   JOB3_COMM
0084' 3403          99            MOVE.W  D3,D2
0086' 7008          100           MOVEQ   #MT_PRIOR,D0
0088' 4E41          101           TRAP    #1
008A' 4E75          102           RTS
                    103
008C' 3478 0112     104 JOB3_COMM MOVE.W  CA_GTINT,A2
0090' 4E92          105           JSR     (A2)
0092' 6618          106           BNE.S   EXIT_JOB3
0094' 70F1          107           MOVEQ   #ERR_BP,D0
0096' 0C43 0003     108           CMPI.W  #3,D3
009A' 6610          109           BNE.S   EXIT_JOB3
009C' 3236 9802     110           MOVE.W  2(A6,A1.L),D1
00A0' 4841          111           SWAP    D1
00A2' 3236 9800     112           MOVE.W  0(A6,A1.L),D1
00A6' 3636 9804     113           MOVE.W  4(A6,A1.L),D3
00AA' 4E75          114           RTS
00AC' 221F          115 EXIT_JOB3 MOVE.L  (A7)+,D1
00AE' 4E75          116           RTS
                    117
00B0' 3478 0112     118 AJOB_PROC MOVE.W  CA_GTINT,A2
00B4' 4E92          119           JSR     (A2)
00B6' 661E          120           BNE.S   EXIT_AJOB
00B8' 70F1          121           MOVEQ   #ERR_BP,D0
00BA' 0C43 0004     122           CMPI.W  #4,D3
00BE' 6616          123           BNE.S   EXIT_AJOB
00C0' 3236 9802     124           MOVE.W  2(A6,A1.L),D1
00C4' 4841          125           SWAP    D1
00C6' 3236 9800     126           MOVE.W  0(A6,A1.L),D1
00CA' 3436 9804     127           MOVE.W  4(A6,A1.L),D2
00CE' 3636 9806     128           MOVE.W  6(A6,A1.L),D3
00D2' 700A          129           MOVEQ   #MT_ACTIV,D0
00D4' 4E41          130           TRAP    #1
00D6' 4E75          131 EXIT_AJOB RTS
                    132
00D8' 3478 0112     133 JOBS_PROC MOVE.W  CA_GTINT,A2
00DC' 4E92          134           JSR     (A2)
00DE' 66F6          135           BNE.S   EXIT_AJOB
00E0' 70F1          136           MOVEQ   #ERR_BP,D0
00E2' 0C43 0001     137           CMPI.W  #1,D3
00E6' 66EE          138           BNE.S   EXIT_AJOB
00E8' 3036 9800     139           MOVE.W  0(A6,A1.L),D0
00EC' C0FC 0028     140           MULU    #CH_LENCH,D0
00F0' 246E 0030     141           MOVE.L  BV_CHBAS(A6),A2
00F4' D5C0          142           ADDA.L  D0,A2
00F6' 2076 A800     143           MOVE.L  0(A6,A2.L),A0
00FA' 43FA 00AC     144           LEA.L   JOB_MESS,A1
00FE' 3478 00D0     145           MOVE.W  UT_MTEXT,A2
0102' 4E92          146           JSR     (A2)
0104' 2448          147           MOVE.L  A0,A2
0106' 7200          148           MOVEQ   #0,D1
0108' 6114          149 GET_JINF  BSR.S   PRT_NJOB
010A' 7400          150           MOVEQ   #0,D2
010C' 7002          151           MOVEQ   #MT_JINF,D0
010E' 4E41          152           TRAP    #1
0110' 4A80          153           TST.L   D0
0112' 6606          154           BNE.S   OUT_JOB
0114' 6132          155           BSR.S   PRT_JINF
0116' 4A81          156           TST.L   D1
0118' 66EE          157           BNE.S   GET_JINF
011A' 7000          158 OUT_JOB   MOVEQ   #0,D0
011C' 4E75          159           RTS
                    160
011E' 48E7 4020     161 PRT_NJOB  MOVEM.L D1/A2,-(A7)
0122' 204A          162           MOVE.L  A2,A0
0124' 3478 00CE     163           MOVE.W  UT_MINT,A2
0128' 4E92          164           JSR     (A2)
012A' 7011          165           MOVEQ   #SD_TAB,D0
012C' 76FF          166           MOVEQ   #-1,D3
012E' 7205          167           MOVEQ   #5,D1
0130' 4E43          168           TRAP    #3
0132' 3217          169           MOVE.W  (A7),D1
0134' 3478 00CE     170           MOVE.W  UT_MINT,A2
```

in D2, so it does a MOVE.W before loading D0 with MY_PRIOR and executing the trap.

AJOB needs four parameters and these are again collected by CA_STINT and put into registers D1 (the job ID), D2 (the new job's priority) and D3 (the activation priority). D0 is then loaded with MT_ACTIV and the trap is executed.

The next procedure is the most complicated of the lot as it needs to scan all the jobs in the machine and find out anything it can about them. The manager trap MT_JINF helps us here. This routine needs the top-of-the-tree job in D2 and the job being scanned in D1. The first thing the routine does is collect its argument and use it to get the ID of the basic # channel into A0. The string 'Job Tag Owner Priority Name' is then printed to that channel and the channel ID is kept in A2 for safety. At this stage D1 will hold the ID of the job currently being scanned, so the routine PRT_NJOB is called to print out its job number and tag. D2 is then cleared, as the SuperBasic interpreter is always on top of the tree (job 0, tag 0) and then the MT_JINF trap is executed. This returns with the ID of the next job in the tree in D1, the owner job ID in D2, the job's priority in the lowest significant byte of D3, its suspension status in the highest significant byte (negative if suspended) and the base address of the job in A0. This information is processed and printed by the PRT_JINF subroutine, and then the loop is repeated unless D1 is equal to zero, which means that the end of the tree has been reached.

```
Job  Tag  Owner Priority Name
0    0    0     32
1    0    0     1        A_CLOCK
2    1    1     1        ALARM
3    2    0     81       SPOOL_1
4    3    0     32       Editor
```

Figure 1. Typical output from procedure described.

PRT_NJOB preserves D1 and A2 on the stack and then uses the UT_MINT utility routine to print the integer in the D1 to the specified BASIC channel. UT_MINT uses only the lowest significant word of D1, which is in this case the job number, so that's printed out first. SD_TAB is then used to move the cursor along a few columns and then the highest significant word of D1 (the job tag) is collected and printed as an integer, again using UT_MINT. The cursor is then moved along again, the registers are retrieved from the stack, and the routine returns.

PRT_JINF has to print rather more information, and preserves all of D1, D2, D3, A0 and A2 on the stack before it does it. It first prints out the lowest significant word of D2 as an integer (the owner job's job number) and then it moves the cursor using SD_TAB as before. We now check the highest significant word of D3 (on the stack!). If this is negative then an 'S' is printed (for 'suspended'), otherwise a space is sent out. Next we get the lowest significant word of D3 from the stack and AND it with 255 ($FF) to leave us with the job's priority in D1. This is printed out and the cursor is moved to the next field using SD_TAB. The next trick is to find the job's starting address on the stack (this was held in A0 on entry to the subroutine). Having loaded this into A2 we now check out the word starting at byte 6 of the job. If this is $4AFB then the job is considered to be in standard QDOS format, which means that the job name must follow the $4AFB word. If so, then it is printed using UT_MTEXT. In either case, a terminating line feed is sent as the last character, the registers are retrieved from the stack and the routine ends. **Figure 1** shows a typical output from this procedure.

The CJOB$ function is also rather complicated, but we know that it will always return a hexadecimal string of 16 characters, so we can define this as a constant using EQU. This function also has a variable number of parameters, so CA_GTLIN is used to collect them as long integers. On return from this routine D3 holds the number of arguments collected. If this is two then A2 must be cleared to zero, which is most easily done by subtracting it from itself (SUBA.L A2,A2). If there are three parameters then all three are collected and put into the registers A2 (start address), D2 (code length) and D3 (data length). D1 is then loaded with –1 to indicate that the job is to be created as a child of the current job,

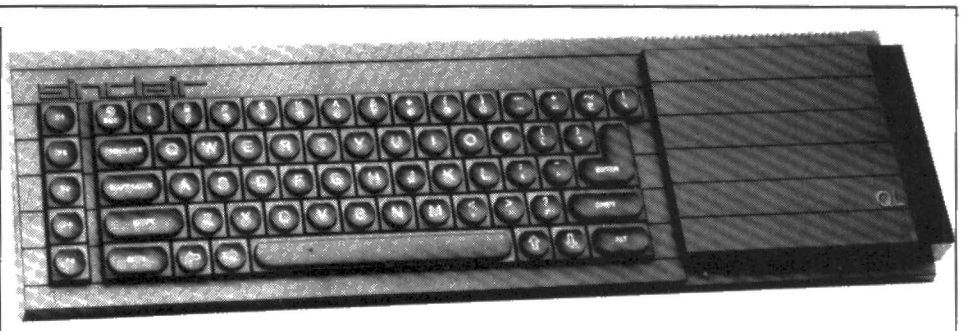
LISTING 1 – Continued

```
013A  4E92          171            JSR     (A2)
013C  7011          172            MOVEQ   #SD_TAB,D0
013E  76FF          173            MOVEQ   #-1,D3
0140  720A          174            MOVEQ   #10,D1
0142  4E43          175            TRAP    #3
0144  4CDF 0402     176            MOVEM.L (A7)+,D1/A2
0148  4E75          177            RTS
                    178
014A  48E7 70A0     179 PRT_JINF   MOVEM.L D1-D3/A0/A2,-(A7)
014E  3202          180            MOVE.W  D2,D1
0150  204A          181            MOVE.L  A2,A0
0152  3478 00CE     182            MOVE.W  UT_MINT,A2
0156  4E92          183            JSR     (A2)
0158  7011          184            MOVEQ   #SD_TAB,D0
015A  76FF          185            MOVEQ   #-1,D3
015C  720F          186            MOVEQ   #15,D1
015E  4E43          187            TRAP    #3
0160  7253          188            MOVEQ   #'S',D1
0162  342F 0008     189            MOVE.W  8(A7),D2
0166  6B02          190            BMI.S   YES_SUSP
0168  7220          191            MOVEQ   #' ',D1
016A  7005          192 YES_SUSP   MOVEQ   #IO_SBYTE,D0
016C  76FF          193            MOVEQ   #-1,D3
016E  4E43          194            TRAP    #3
0170  322F 000A     195            MOVE.W  10(A7),D1
0174  0241 00FF     196            ANDI.W  #$FF,D1
0178  3478 00CE     197            MOVE.W  UT_MINT,A2
017C  4E92          198            JSR     (A2)
017E  7011          199            MOVEQ   #SD_TAB,D0
0180  76FF          200            MOVEQ   #-1,D3
0182  7219          201            MOVEQ   #25,D1
0184  4E43          202            TRAP    #3
0186  246F 000C     203            MOVE.L  12(A7),A2
018A  0C6A 4AFB 0006 204            CMPI.W  #$4AFB,6(A2)
0190  660A          205            BNE.S   NOT_STDF
0192  43EA 0008     206            LEA.L   8(A2),A1
0196  3478 00D0     207            MOVE.W  UT_MTEXT,A2
019A  4E92          208            JSR     (A2)
019C  720A          209 NOT_STDF   MOVEQ   #10,D1
019E  76FF          210            MOVEQ   #-1,D3
01A0  7005          211            MOVEQ   #IO_SBYTE,D0
01A2  4E43          212            TRAP    #3
01A4  4CDF 050E     213            MOVEM.L (A7)+,D1-D3/A0/A2
01A6  4E75          214            RTS
                    215
01A8  001E          216 JOB_MESS   DC.W    30
01AA  4A6F 6220 2054 6167 217      DC.B    'Job  Tag  Owner Priority Name',10
      2020 4F77 6E65 7220
      5072 696F 7269 7479
      204E 616D 650A
                    218
                    219 * A function to create a job and return the tag,
                        job number and base
                    220 * address as one long hex string
                    221
      =0010         222 ANS_LEN    EQU     16              Length of return string
                    223
01CA  3478 0118     224 FN_CJOB    MOVE.W  CA_GTLIN,A2
01CC  4E92          225            JSR     (A2)
01CE  660E          226            BNE.S   EXIT_CJOB
01D0  0C43 0002     227            CMPI.W  #2,D3
01D4  670A          228            BEQ.S   TWO_PARAM
01D6  0C43 0003     229            CMPI.W  #3,D3
01DA  6708          230            BEQ.S   THREE_PA
01DC  70F1          231            MOVEQ   #ERR_BP,D0
01DE  4E75          232 EXIT_CJOB  RTS
                    233
01E0  95CA          234 TWO_PARAM  SUBA.L  A2,A2
01E2  6004          235            BRA.S   GET_PARAM
01E4  2476 9808     236 THREE_PA   MOVE.L  8(A6,A1.L),A2
01E8  2436 9800     237 GET_PARAM  MOVE.L  0(A6,A1.L),D2
01EC  2636 9804     238            MOVE.L  4(A6,A1.L),D3
01F0  72FF          239            MOVEQ   #-1,D1
01F2  5089          240            ADDQ.L  #8,A1
01F4  0C43 0003     241            CMPI.W  #3,D3
01F8  6602          242            BNE.S   NOT_THREE
01FA  5889          243            ADDQ.L  #4,A1
01FC  2F09          244 NOT_THREE  MOVE.L  A1,-(A7)
01FE  224A          245            MOVEA.L A2,A1
0200  7001          246            MOVEQ   #MT_CJOB,D0
0202  4E41          247            TRAP    #1
0204  4A80          248            TST.L   D0
0206  66D6          249            BNE.S   EXIT_CJOB
0208  2801          250            MOVE.L  D1,D4
020A  2A08          251            MOVE.L  A0,D5
020C  225F          252            MOVE.L  (A7)+,A1
```

and A1 is reset and saved on the stack. The MT_CJOB trap is then used to physically create the job, and D0 is tested to see if there was an error during creation – the most likely being 'Out of memory'. If there was an error then the routine returns to basic straight away and reports the error.

If everything has been okay so far, D1 will hold the ID of the newly created job and A0 will hold its starting address. We need to convert all these to hex and return them as a string to SuperBASIC as the function result. We first use BV_CHRIX to get 18 bytes of RI stack space (a 16 character string occupies 18 bytes – 16 for the characters and 2 for the length). We then stack the length of the return strong on the RI stack and store the RI stack pointer in BV_RIP ready for the return to Basic.

The TQ_HEX subroutine converts the number held in D1 to an eight character hex string, putting it into the buffer pointed to by A0, so by loading A0 with an effective address of 2(A6,A1.L) we can ensure that the string is stored on the RI stack as it is created. Having done that we put the return type (string) in D4, clear D0 and return to Basic.



The TQ_HEX routine very simply isolates each nibble of D1, converts it into the corresponding hex digit, and stores it in the address pointed to by A0, which is then incremented ready for the next character.

Having written these procedures they are included in the SuperBASIC name list in the normal way by loading the code into the resident procedure area and calling the starting address:

```
address=RESPR(code_length)
CALL address
```

All we need to discuss in the QL multitasking field now is the PIPE device, job to job communication, and 'getting your hands dirty'.

LISTING 1 – Continued

```
020E' 7212          253          MOVEQ   #18,D1
0210' 3478 011A     254          MOVE.W  BV_CHRIX,A2
0214' 4E92          255          JSR     (A2)
0216' 3DBC 0010 9800 256         MOVE.W  #ANS_LEN,0(A6,A1.L)
021C' 41F6 9802     257          LEA.L   2(A6,A1.L),A0
0220' 2D49 0058     258          MOVE.L  A1,BV_RIP(A6)
0224' 2204          259          MOVE.L  D4,D1
0226' 610A          260          BSR.S   TQ_HEX
0228' 2205          261          MOVE.L  D5,D1
022A' 6106          262          BSR.S   TQ_HEX
022C' 7801          263          MOVEQ   #RET_STR,D4
022E' 7000          264          MOVEQ   #0,D0
0230' 4E75          265          RTS
0232' 7007                                  266
0234' E999          267 TQ_HEX   MOVEQ   #7,D0
0236' 1401          268 HEX_LOOP ROL.L   #4,D1
0238' 02B2 0000 000F 269         MOVE.B  D1,D2
023E' 0C02 000A     270          AND1.L  #$F,D2
0242' 6D02          271          CMPI.B  #10,D2
0244' 5E02          272          BLT.S   HEX_DIG
0246' 0602 0030     273          ADDQ.B  #7,D2
024A' 10C2          274 HEX_DIG  ADD1.B  #'0',D2
024C' 51C8 FFE6     275          MOVE.B  D2,(A0)+
0250' 4E75          276          DBRA    D0,HEX_LOOP
                    277          RTS
                    278                                  279          END
```

BOOK REVIEWS

Paul Beverley examines three books which feature interfacing projects for the BBC micro.

Interfacing the BBC Microcomputer

Colin Opie
McGraw-Hill, £8.95

Here it is at last, a definitive book on interfacing the BBC Microcomputer! Its twelve chapters and six appendices contain a vast amount of technical information. It must represent a mammoth amount of work on Colin Opie's part. The £8.95 you would pay for the book hardly reflects the work involved in writing. I think it must have been done more for love than for the money!

The book is divided into three parts. In the first section the author deals with basic concepts, starting very simply but very quickly building up to such a high rate of information output that all but the most experienced person would quickly get left behind. Never fear, there is an excellent bibliography at the end of each of two of the first three chapters.

The second section deals in more detail with programming the various interfaces on the BBC microcomputer. The longest chapter in this section is, not surprisingly, the one which deals with the user port and its associated 6522 VIA chip. The other chapters deal with analogue input, light pens, the RS423 and printer ports, and the 1MHz Bus. Throughout this section there are BASIC procedures and functions along with their assembly language equivalents, together with examples of how they can be included in programs.

After the 135 pages of very intensive theory which makes up the first two sections, we come at last, in the final 30 pages, to some interfacing project ideas for the reader to try. They are based on a series of inter-

face boards with circuits designed by the author and PCB layouts by Watford Electronics from whom full kits of parts should eventually become available. At the time of writing, however, Watford Electronics are not able to say how much the kits will cost or when they will be available. All the circuit diagrams and PCB layouts are given so that if Watford do not produce the kits, you can build your own units. I know you can't have everything, but I would like to have seen more practical projects. I suspect that the cost of the motherboard system may be such that unless you can see more ways in which you can use the system, you will not feel that it justifies the financial outlay.

In terms of information per pound, this book represents very good value. Technically it is very sound and I have only picked up a couple of very small technical and typographical errors. However this is not a book for beginners. It is more akin to those weighty tomes used in universities which are usually entitled, "An Introduction to . . .", or something equally inappropriate. If you have watched "Computers in Control" on TV and think you'd like to "have a go" yourself, then I would not advise you to start with Colin Opie's book. You will be submerged in technicalities within five minutes. If on the other hand you have done a little bit of interfacing already, then this book *is* for you. I personally will find parts of this book extremely valuable for reference purposes.

BBC Hardware Projects

Don Thomasson
Melbourne House, £8.95

This second book also has a mixture of theory and practice, but the balance swings in favour of the projects which occupy 130 of the book's 180 pages. In fact this section is not written by the titled author, Don Thomasson. The acknowledgements page says that Fabian Stretton wrote, designed and developed all the projects.

The theory section gives some specific information about each of the BBC interfaces with suggestions as to the sorts of applications for which they might be used. The section finishes with some details about interrupts and two chapters about control theory. It is difficult to see at what level the author intends to pitch the book. The information varies in depth from vague generalities that a beginner could easily understand to some obscure technical facts which I did not really understand myself.

I suspect that Mr. Thomasson may never have actually used some of these interfaces which he talks about. For example, when dealing with the analogue to digital converter he refers to the reference voltage and talks about "1.8635 volts, though this will vary somewhat with temperature". In fact the reference voltage varies by about 6% as the computer warms up! He also says that half a millivolt of noise pickup could change the value returned by the converter. He seems unaware that most of these ADC chips have at least 40 millivolts of noise on the analogue earth line.

My overall impression of the projects is one of a series of academic exercises not really geared to the

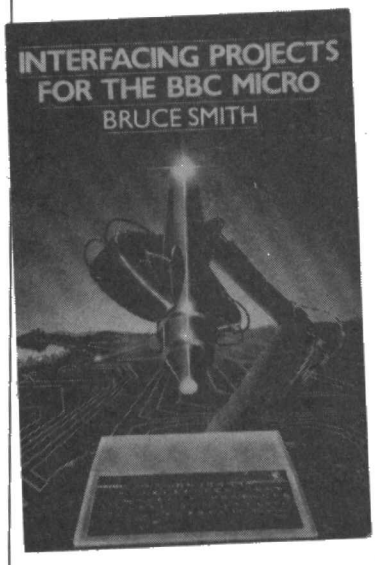
average microcomputer owner who is interest in doing some interfacing. It tells you for example "how to use your BBC to operate up to two hundred and fifty-five mains powered appliances and domestic lighting", but the logic circuits to do this would cost over £40, even before adding any driver circuits. Then there are three different ways of making a hexadecimal keypad, but the only software provided is a simple BASIC program for each piece of hardware which prints out the hexadecimal value of the key being pressed. Then there is a digital to analogue converter made up of 25 resistors plus 6 IC's costing probably slightly more than the single chip ZN425 which would do the same job. What about the idea of hacking up a "binary counting display" using eight 100 mA incandescent lamps rated at anywhere between 5 and 22 volts! What is wrong with eight resistors and eight LED's on the printer port?

To be fair though, there are a number of projects which are useful. There are sections on light and temperature measurements and a design for a light pen which looks quite reasonable and there is a very interesting section on position sensing and motion measurement.

One general criticism is that there is really very little software provided which does more than just test whether or not the hardware is working. Also, the programs that are given are almost totally unstructured. Many of them are reminiscent of the proverbial "GOTO Show". There are very few functions or procedures which you could transfer into your own programs.

The most serious technical criticism though is that the book talks about using the computer to measure voltages up to 2,000 volts dc and 1,400 volts ac. It does warn you of the danger of doing so and it provides you with a protection circuit. But they suggest that it could be built on 0.1" Veroboard! If you examine the specification of this material you will find that it has a breakdown voltage of 1,200 volts (did the author actually test it up to the 2,000 volts that he suggests using it for!), and the maximum recommended working voltage is only 400 volts. Secondly, the Veroboard layout given for an auto ranging voltmeter has the 1,000 volts input line on a track which is immediately adjacent to the track connected to the ADC input of the computer. All it needs is a greasy thumb-print, a solder fleck or a stray wisp of copper from a track break, and you will get an arc between the two tracks and you have 1,000 volts applied to the inside of your BBC micro. Finally the fixing screws on two of the Veroboard circuits are centred just one track away from the high voltage input line. Surely one would expect projects in a book of this nature to be engineered to higher standards than this, especially as regards safety?

Not a book for beginners.



Interfacing Projects for the BBC Microcomputer
Bruce Smith
Addison Wesley, £6.95

This is the most practically oriented of these three books. It contains a number of projects which are potentially useful. For example an EPROM programmer, s sophisticated looking XY plotter and the inevitable light pen. The software is much more comprehensive and is thankfully, to some extent, structured. The author even uses indentation in the listings to show the structure!

However I have similar technical reservations about this book as with the previous one. In the chapter on the light controller, the author suggests that you use Veroboard for circuits which control mains devices. Admittedly he uses a more careful layout to try to ensure isolation, and the mains circuits are optically isolated. Nevertheless I would not be happy to have mains voltages on the same piece of Veroboard as a circuit connected directly into my computer. I am also worried about the interconnections for the light-controller. It needs to be connected to the home-made power supply (chapter 5), the D to A converter (chapter 13) and the computer's own auxilliary power supply output. There is no diagram to show how the different boards are interconnected, just labels on the controller board like "+12 volts to +15 volts from BR1". BR1 is on the power supply board, but nowhere is it marked with "12 to 15 volts". It is not at all easy to work out how to connect it up, and only one wire in the wrong place could have very serious consequences.

In addition there are a number of technical and typographical errors. Some are not too serious but one or two are potentially dangerous. For example the design given for a 1 to 30 volt, 3 amp power supply is quoted as using a 1 amp bridge rectifier and a 50 VA transformer (to give up to 90 watts output?). The transformer voltage is only 24 volts rms (= 34 volts peak at no load) which is hardly sufficient to give 30 volts output from the regulator. It has a 1,000 uF smoothign capacitor which, according to my calculations would give roughly 7 volts per amp ripple, ie 21 volts ripple at 3 amps. It makes one wonder whether the design has actually been tested at the full output.

The most serious mistake of all is that the two electrolytic capacitors are quoted as 6.3 volts working. Now anyone with any experience in electronics will realise the mistake, but a beginner may not. If you put the 34 volts peak coming out of the rectifiers across a 6.3 volt 1,000 uF capacitor, you will get a "component failure"! Indeed if you accidently put the capacitor into the circuit the wrong way round (which a beginner might easily do as there is no written warning) then I can almost guarantee that the capacitor would explode. These errors may all be typographical, but they do not inspire confidence in the technical standard of the book as a whole and, at the time of writing, there is no erratum slip available either with the book or from the publishers.

Once again, this is not, in my view, a book for beginners although it has some good ideas for the more experienced person.

Our reviewer, Paul Beverley, is the author of the new Service Manual for the BBC micro.

INTERACTIVE VIDEO

Following last month's feature, we bring news of more developments in this exciting medium.

Following the recent publicity on interactive video technology comes news of two developments in the field which illustrate the versatility and potential importance of the subject.

The BBC has announced that, in association with Acorn Video and Philips, it will over the next eighteen months collect and collate more than two million pages of text, and thousands of maps and pictures to be stored on video-disc as a permanent record of Britain in the 1980s.

The Domesday Project was dreamt up as a way of marking the 900th anniversary of the original Domesday Book of 1086, but it was not possible to carry out the largest survey of a country ever undertaken without the help of high tech. No medium other than video-disc could store information amounting to two complete sets of the Encyclopaedia Britannica so economically, and it is intended to market Domesday as two double-sided discs in the Autumn of 1986.

Hardware options

To play the discs, however, the hardware must be not just available, but affordable. At the moment, the options are between buying a system from Acorn Video, consisting of a BBC micro, monitor, genlock board, floppy disc drive, interfacing software and driving software, plus an optical disc player, for the princely sum of about £3,000 or doing it yourself. It is possible to put together a system, using two monitors and some ingenuity, but it would mean that the micro could not be used for anything else without dismantling everything, and it would still not be cost effective.

Philips and Acorn Video are working to develop a dedicated optical disc system which will be marketed as a BBC video disc player, much as Acorn's machine has been sold as the BBC micro. However, although Domesday is the catalyst for pushing existing technology beyond its present limits, the final result is still expected to cost between £1100 and £1500.

Domesday will use the optical discs, specifically Philips' Laservision version, since the project is dealing with long-term storage and frequent access of data. For this kind of application, the durability of optical discs is necessary since the disc is played by a laser beam, there is no physical contact made with the disc and little risk of wear over a long period.

Meanwhile, back in the Orient, the Japanese are also becoming interactive. For the modest sum of 9,800 yen (about £30) you can now buy, from JVC, three interactive programs for your VHD video disc system, powered by "any type of home computer". Admittedly, you have to live in Japan to cash in on this, but JVC (UK) Limited will certainly have plans to introduce "Vroom", "The Player's Club" and "Alice in Chemical Reaction" into Britain, probably in the wake of the MSX invasion.

The games are on VHD video discs, and the interactive element is laid down directly onto the disc. Standardisation has, the company claims, been achieved using a VHD standard language and an interpreter which enables any brand of micro to be used. But there is no information yet on how much the VHD hardware will cost. With a basic VHD player retailing for around £199, the whole concept is likely to collapse if the interactive version costs more than £200 or so. It's unlikely that the purchaser of a £200 or £300 micro will relish the idea of paying out massive sums in order to play yet one more type of game.

In Britain, Thorn EMI are actively advertising their AHD video disc system, which uses a VHD player and which the company claims will hold pictures, sound and computer data on disc. Whether their product will run JVC's "Interaction" series remains to be seen, but one thing is clear. If the number of products available continues to proliferate, interactive video could become a dead issue, buried in a morass of claims, counter-claims and publicity hype. It happened with video disc players in America in the late seventies. It could happen again with the infinitely more exciting interactive version.

FREE EVERY MONTH

Your ROBOT

BRITAIN'S FIRST ROBOTICS MAGAZINE

Each month, Your Robot reports on the latest news from the world of low cost robotics. In addition in-depth features explain the theory behind the operation of robots and computer control systems.

REMCON TEACH-ROBOT

Last month we announced that Your Robot was to describe the construction of a versatile robot arm. Detailed description of putting the Remcon arm together begins next month – in this issue we describe the various features offered by the teach-robot.

The teach-robot arm has six axes, movement of which is controlled by DC motors equipped with a feedback system based on a slotted disc and a photo emitter/detector combination. Movement in the four major axes is accomplished by an arrangement in which the motor drives a piston that is anchored to a suitable point on the assembly. Wrist rotation and gripper motion is achieved by worm gears mounted on the two remaining DC motors.

DESCRIPTION OF FUNCTIONS

The body is carried by a main bearing mounted on the base plate. The associated drive gives angular movement of 90 degrees. The shoulder, arm and wrist drives also produce movements of 90 degrees via their lead screw.

The shoulder, which is swivelled by 90° vertically at the bearing point in the body part carries the arm. The arm is lifted or lowered by 90° at the common centre point of the shoulder and the arm. The hand, swivel mounted in the front part of the arm, contains a drive to open and close the fingers and a drive, which turns around the whole hand by more than 360°, along the longitudinal axis of the fingers. A cable harness carries all electrical connections to the 25-pin connector.

The arm is constructed with careful attention to achieving a rigid structure with the minimum of slack. This is important if positional accuracy of the gripper is to be maintained during operation.

Next month we shall provide detailed notes on construction of the arm together with full details of the robot kit to be supplied by Remcon.

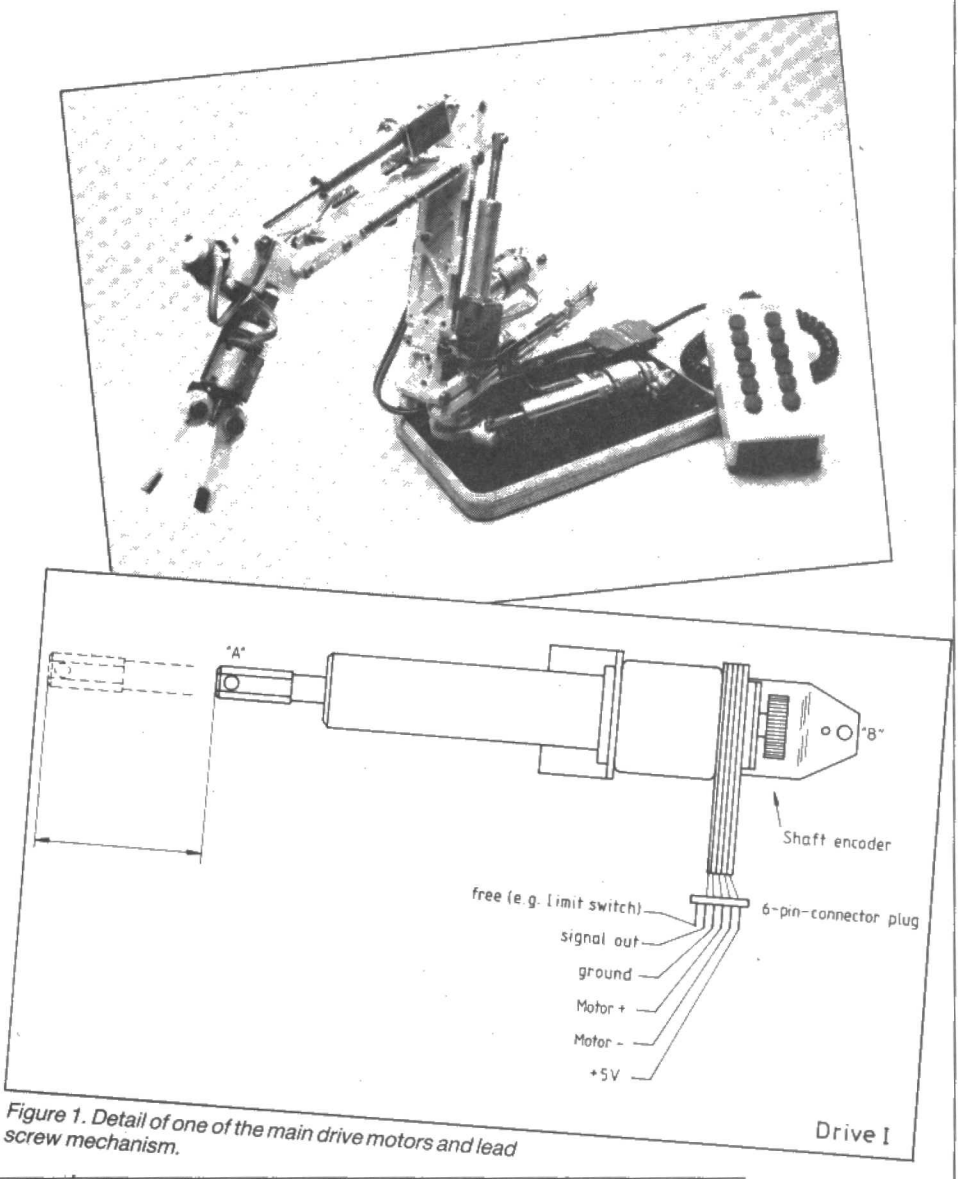
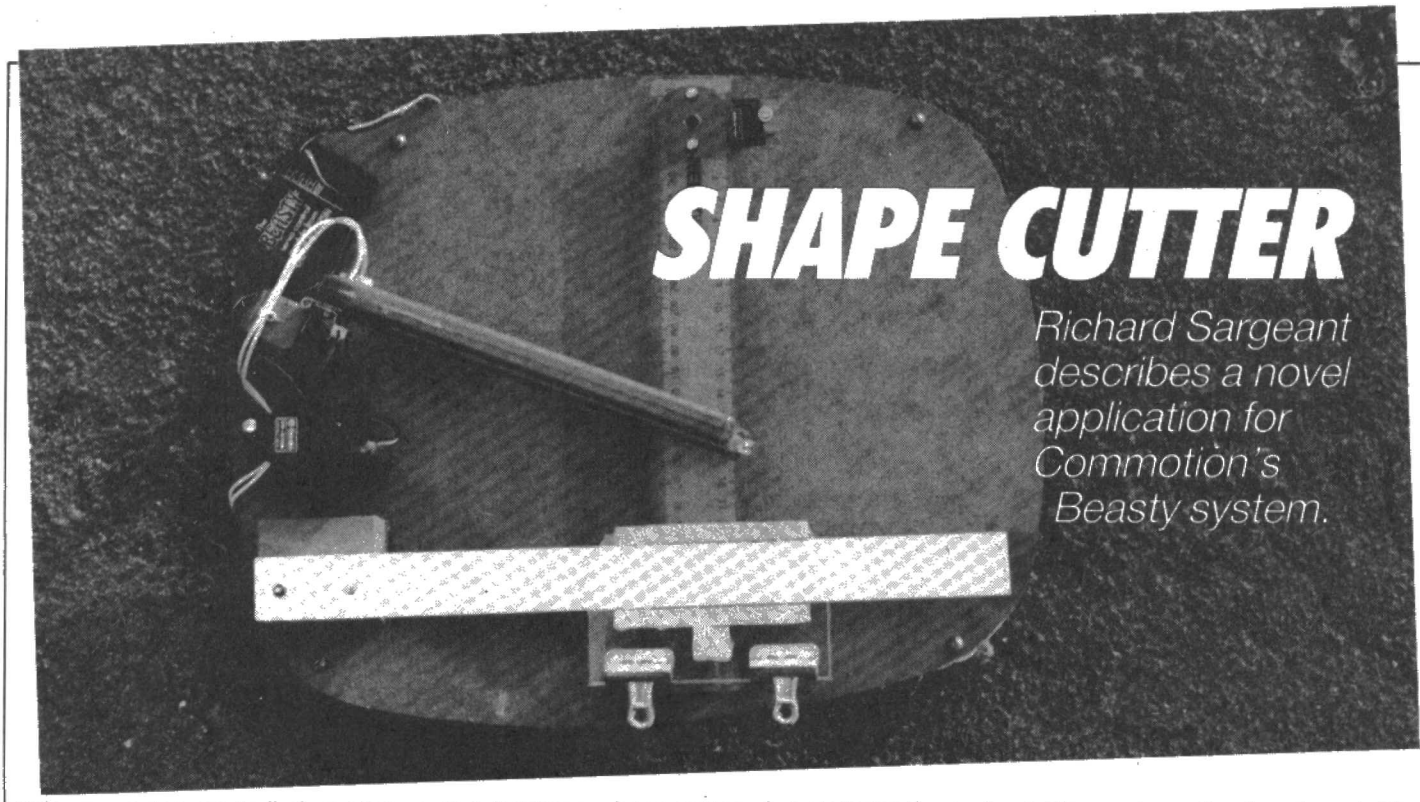


Figure 1. Detail of one of the main drive motors and lead screw mechanism.

Your Robot, 155 Farringdon Road, London EC1R 3AD. **Editor:** Gary Evans, **Deputy Editor:** William Owen, **Production Editor:** Liz Gregory, **Advertisement Manager:** Richard Jansz, **Advertisement Production:** Yvonne Moyser.



SHAPE CUTTER

Richard Sargeant describes a novel application for Commotion's Beasty system.

This computer controlled system cuts shapes with a "hot wire" tool, using a method so simple that it is possible to transform the standard 12" x 12" polystyrene ceiling tile into a 64 piece jigsaw puzzle in about 2 minutes. What we are going to do, with the aid of a robotic device called "Beasty", is construct a computer-controlled polystyrene tile cutter which will cut shapes from patterns stored in the computer's memory. As such we are duplicating, on a very small scale, the automatic milling machines used by most modern engineering firms today. And, just to be seasonal, we shall use the Beasty Cutter to mass-produce some polystyrene garlands, stars, and other Yuletide Novelties.

CONSTRUCTION

The tile cutter works as a highly specialised X-Y plotter, with the hot wire taking the place of the drawing pen. The tile itself moves along the X axis and the hot-wire moves along the Y axis. The whole construction is mounted on a baseboard about one foot square (see **Figure 1**). Two powerful Futaba servo motors provide the motive power, and are driven in turn by the Beasty, of which more will be said later. The whole rig takes about one evening to put together, but there are some rather strange components on the shopping list which you should make sure you have before you start.

The hand-held hot-wire may be difficult to obtain but, if that's the case, you can make your own. Try wallpaper shops, hardware shops and market stalls for the tool, which is made of plastic (invariably red plastic), and looks like a giant U-shaped magnet. It runs on a 4.5V battery via a built-in push button switch. If you want to make your own, then a 2" piece of resistance wire held taut between two parallel rulers (wooden or plastic) will do the trick. School physics labs are good places to try for this odd length of wire, or you might try unwinding some wire from an old relay.

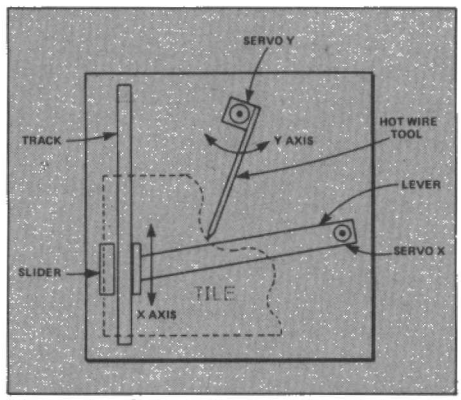


Figure 1. Construction is mounted on a baseboard.

THE X AXIS

Readers should not feel obliged to follow to the letter the form of construction adopted for the prototype – basically you should use whatever materials come to hand! The polystyrene tile is attached to a slider, and held there securely by two bulldog clips. The slider runs freely on a track, which is about 23mm above the baseboard. The track is a straight smooth piece of curtain rail, 330mm in length. The slider, which straddles the track, is made up of three or four different pieces, marked A, B, C & D in the cross-section view, **(Figure 2).** Piece A is the top plate, which is the flat clear part of a standard audio-cassette case. Pieces B are tubular pieces of plastic, obtainable from model shops, or equally suitable, hexagonally-sectioned biro cases. This is a low-cost project! Piece C is a plastic channel – the sort sliding doors glide in. Packing pieces D may also be required, and these are cardboard or thin plastic. The whole arrangement is glued together with model-maker's plastic glue or an impact adhesive. Test that the slider runs smoothly on the track. One small block of wood at one end of the track raises it the required 23mm off the surface of the baseboard.

A servo motor, recessed into the baseboard and on the opposite side to the track, is connected to the slider by means of a long lever. The lever is about 280mm in length, and is made of plastic – in fact, it is a ruler, or *the* ruler if you've already finished marking out the baseboard. One end is bolted to the circular "horn" which is supplied with the servo motor. This puts the lever about 10mm above the surface of the baseboard. The other end of the lever has a stub-foot or glider on its undersurface, and a stub-upright on its upper surface. The former is made of a drawing pin while the latter is an 8 BA bolt to which a metal sleeve has been added. As the servo turns, so the foot describes an arc across the surface of the baseboard. The stub-upright follows the same path but because it fits into the plastic channel of the slider it moves the slider too, and that follows the straight path of the track.

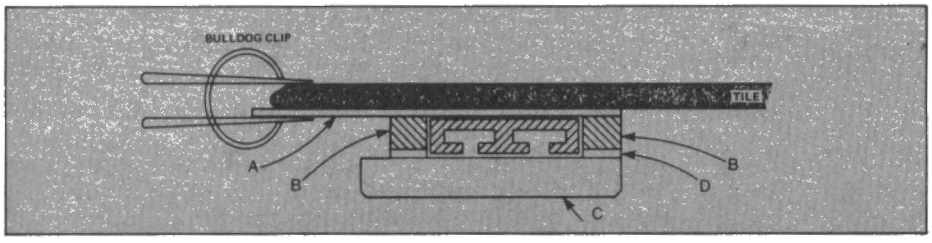


Figure 2. Cross section view of slider.

THE Y AXIS

The Y axis is merely the pivoted hot-wire. It should be mounted directly upon the circular horn of the second servo motor, using a

soft-aluminium bracket. This method gives the greatest accuracy to the cutting action. The prototype used the commercial U-shaped tool already described; it was allowed to pivot on a hinge attached to a wooden upright and the servo motor was mounted some 60mm distant and moved the tool by means of a wire link. Such a link introduces play into the system, and reduces the accuracy of the cut shapes.

The lower arm of the hot-wire tool clears the baseboard by 15cm. Its on/off button is either electrically disabled by shorting out, or physically glued permanently on.

"... duplicates, on a small scale, an automatic milling machine".

THE BEASTY

The Beasty is a little black box of tricks which enables 1,2,3 or 4 servo motors to be controlled by one TTL output line. It comes with its own software, and an explanatory manual called the Beasty Trainer's Handbook. Beasties work with the BBC computer, although they are being made available for other machines as well. With the BBC computer, operation is simplicity itself. The little black Beasty box takes its signal from bit 7 of the user port, and power for itself and the 2 servos from the BBC auxillary power supply. All the leads and connectors are supplied ready made.

SETTING UP

Since the servo motors are powerful, they should be set working correctly without a load on their axles so there is no possibility of them swinging the equipment into odd positions and thus doing damage. Unscrew the circular horns from the servos, thus removing all levers. Driving the servos from BASIC should be done in the following manner:

```
*LOAD DRIVER 2800
DRIVER=&2800
CALL DRIVER
```

Beasty's driving routine loaded and initialised.

```
X%=0          Address servo "0"
Y%=126        Set an angular value
CALL DRIVER+3 Execute servo
              command
```

Observe which servo moves.

```
X%=255        Command address
Y%=0          Specifies "servo off"
CALL DRIVER+3 Execute special
              command
```

Physically change the servo leads if necessary so that the servo which moved becomes the X axis servo. Now using X%=1 (which is servo "1") and again Y%=126 and CALL DRIVER+3, energise the Y axis servo. The axles of both servos have taken up their mid-point positions, and you should screw the horns and levers back on in their *middle-of-baseboard positions*. Now write a short BASIC program which alters the value of Y% both upwards and downwards from 126 until you have found the values corresponding to the physical limits of the cutting board. Using more BASIC or Beasty's ROBOL language, you can now begin cutting your polystyrene shapes. Happy Christmas!

Footnote For computers not lucky enough to be able to run the Beasty, we hope to explain how to run servo motors in a forthcoming issue of **Your Robot.**

PARTS LIST

2 Futaba FP-S128 servo motors
1 Beasty (includes software)
12" approx curtain rail
12" plastic ruler
Square foot hardboard baseboard
Plastic channel etc as specified
Hot-wire tile cutter
Glue, screws, scrap of wood.

The Beasty is available from *Commotion, 241 Green Street, Enfield EN3 7SJ.*

HUNDREDS OF NEW LINES

The amazing Maplin Catalogue is here again! The new edition is packed with hundreds and hundreds of new electronic components to bring you right up to date with all the latest developments. As all home constructors agree (and a good many professionals too) the Maplin Catalogue is the one essential piece of equipment they really need. And now with all our **prices on the page** the Maplin Catalogue is better value than ever.

On Sale From 10th November 1984.

Pick up a copy as soon as it's published at any branch of W.H. Smith or in one of our shops. The price is still just £1.35, or £1.75 by post from our Rayleigh address (quote CA02C).

Post this coupon now for your copy of the 1985 catalogue.
Price £1.35 + 40p post and packing. If you live outside the U.K. send £2.40 or 11 International Reply Coupons.
I enclose £1.75.

Name .....

Address .....

.....

.....

MAPLIN
ELECTRONIC SUPPLIES LTD

Maplin Electronic Supplies Ltd. Mail Order: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Tel: Southend (0702) 552911. •Shops at: 159-161 King Street, Hammersmith, London W6. Tel: 01-748 0926. •8 Oxford Road, Manchester. Tel: 061-236 0281.
•Lynton Square, Perry Barr, Birmingham. Tel: 021-356 7292.
•282-284 London Road, Westcliff-on-Sea, Essex. Tel: 0702-554000.
•46-48 Bevois Valley Road, Southampton. Tel: 0703-25831. All shops closed all day Monday.